

KONZEPTION UND UMSETZUNG
EINER DOKUMENTATION FÜR DIE
SOFTWARE „WPS-MANAGER 5.0“
UNTER VERWENDUNG VON XML-
TECHNOLOGIEN

MASTERTHESIS

IM FACHBEREICH INFORMATIK UND
KOMMUNIKATIONSSYSTEME DER
HOCHSCHULE MERSEBURG (FH)

vorgelegt von
Dipl.-Ing. (FH) Patrick Henkel
aus Halle (Saale)

Erster Betreuer:

Dr. Thomas Meinike

Zweiter Betreuer:

Dipl.-Phys. Thomas Vetterling

eingereicht am:

12. Dezember 2011

1	Einleitung.....	1
1.1	Vorwort.....	1
1.2	Zielsetzung.....	2
1.3	Aufbau.....	2
2	Theoretische Grundlagen	3
2.1	Cross-Media-Publishing	3
2.2	XML	3
2.2.1	Aufbau eines XML-Dokuments	6
2.2.2	Elemente, Tags und Attribute	7
2.2.3	Wohlgeformtheit und Gültigkeit.....	8
2.3	Namensräume.....	9
2.4	DocBook-XML	10
2.5	XSL	13
2.6	XSLT	14
2.7	XPath	19
2.7.1	Knotentypen.....	19
2.7.2	Prädikate	21
2.7.3	Achsen	22
2.7.4	Verkürzte Schreibweisen.....	26
2.7.5	Operatoren	28
2.8	XSL-FO	29
2.8.1	Grundgerüst eines XSL-FO-Stylesheets	31
2.8.2	Seitenlayout.....	31
2.8.3	Seitenbereiche	32
2.8.4	Mehrspaltige Satzspiegel	35
2.8.5	Schreibrichtung und Textausrichtung	35
2.8.6	Elementstruktur	37

2.8.7	Inhalt	37
2.8.8	Festlegen von Seitenfolgen	40
2.8.9	Maßeinheiten	42
2.8.10	Farben.....	43
2.8.11	Blockelemente.....	43
2.8.12	Listen	44
2.8.13	Grafiken	47
2.8.14	Hyperlinks	48
2.8.15	Attribut-Sets	49
2.9	Neuerungen in XSLT 2.0 und XPath 2.0.....	50
2.9.1	xsl:result-document	50
2.9.2	xsl:function	52
2.10	Kommandozeilenbasiertes Transformieren.....	54
2.10.1	Vorbereitung.....	54
2.10.2	Händische XSLT-Transformation mit Saxon HE.....	55
2.10.3	Händische FO-Transformation mit Apache FOP.....	55
2.11	Sprachliche Gestaltung von Benutzerhandbüchern.....	56
2.12	Ausgabemedien und Endgeräte	58
2.13	Responsive Webdesign.....	60
3	Praktische Umsetzung	65
3.1	Softwareübersicht	65
3.1.1	XMLmind XML Editor	65
3.1.2	Syntext Serna Free	66
3.1.3	Oxygen XML Editor	67
3.1.4	XML Copy Editor	68
3.1.5	Webocion Scriptly	69
3.2	Verwendete Entwicklungsumgebung.....	70
3.2.1	Silbentrennung in Apache FOP	71

3.2.2	Darstellungsfehler in Apache FOP	71
3.2.3	Benutzerdefinierte Autorenansicht in Oxygen XML Editor	72
3.3	XSL-Stylesheets	75
3.4	Zielformate	75
3.4.1	XHTML	76
3.4.2	PDF	76
3.5	Informationsstruktur	76
3.6	Sprachliche Gestaltung	78
3.6.1	Makrostruktur	79
3.6.2	Mikrostruktur	80
3.7	Layout	81
3.7.1	PDF-Ausgabe	81
3.7.2	XHTML-Ausgabe	85
3.8	Abbildungskonzept	87
3.9	Besonderheiten der XHTML-Ausgabe	88
3.9.1	JavaScript	88
3.9.2	Baumnavigation	88
3.9.3	Download-Link	89
3.9.4	Blättern-Funktion	89
3.9.5	Nach-Oben-Scrollen	90
3.9.6	Breadcrumbs	90
3.9.7	Anpassbare Schriftgröße	90
3.9.8	Verkleinerte Abbildungen	90
3.9.9	Suchfunktion	91

4	Fazit	94
5	Quellen.....	96
5.1	Bücher.....	96
5.2	Zeitschriften	97
5.3	Internet.....	98
5.4	Sonstige	102
5.5	Software.....	103
5.6	Abbildungen	104
6	Anhang	106
7	Eidesstattliche Erklärung.....	107

Abbildung 1: Zusammenhang der Meta-, Auszeichnungs- und Transformations-sprachen	5
Abbildung 2: Aufbau eines Elements	8
Abbildung 3: Prinzip der Erzeugung eines (X)HTML-Dokuments	15
Abbildung 4: Ausgabe des Ergebnisdokuments im Firefox Browser.....	17
Abbildung 5: child-Achse	22
Abbildung 6: parent-Achse	23
Abbildung 7: self-Achse	23
Abbildung 8: ancestor-Achse	23
Abbildung 9: ancestor-or-self-Achse	24
Abbildung 10: descendant-Achse.....	24
Abbildung 11: descendant-or-self-Achse	24
Abbildung 12: preceding-sibling-Achse	25
Abbildung 13: following-sibling-Achse	25
Abbildung 14: preceding-Achse	25
Abbildung 15: following-Achse	26
Abbildung 16: Prinzip der Erzeugung eines PDF-Dokuments	30
Abbildung 17: Seitenbereiche.....	33
Abbildung 18: mögliche Schreibrichtungen und Ausrichtungen.....	36
Abbildung 19: Elementstruktur eines XSL-FO-Stylesheets	37
Abbildung 20: Ausgabe der PDF im Adobe Reader	40
Abbildung 21: Ausgabe formatierter Blockelemente im Adobe Reader	44
Abbildung 22: Ausgabe einer ungeordneten Liste ohne Abstandsdefinition ...	45
Abbildung 23: Ausgabe einer ungeordneten Liste mit Abstandsdefinition	46
Abbildung 24: Layout bei verschiedenen Auflösungen.....	63
Abbildung 25: DocBook-XML-Dokument in XMLmind XML Editor.....	66
Abbildung 26: DocBook-XML-Dokument in Serna Free	67
Abbildung 27: DocBook-XML-Dokument im Oxygen XML Editor.....	68
Abbildung 28: XSL-Stylesheet im XML Copy Editor	69

Abbildung 29: JavaScript-Dokument in Webocton Scriptly.....	70
Abbildung 30: Darstellungsfehler bei aktivierter Silbentrennung	71
Abbildung 31: kein Darstellungsfehler bei deaktivierter Silbentrennung	71
Abbildung 32: unbearbeitete Autorenansicht in Oxygen XML Editor	74
Abbildung 33: bearbeitete Autorenansicht in Oxygen XML Editor	74
Abbildung 34: Layout des Deckblatts	82
Abbildung 35: Layout der Rückseite.....	83
Abbildung 36: Layout der Inhaltsseiten	84
Abbildung 37: Beispielseiten des Zielformats PDF	85
Abbildung 38: Layout der XHTML-Ausgabe	86
Abbildung 39: Beispielseite des Zielformats XHTML.....	87

CSS	Cascading Style Sheets
DTD	Document Type Definition
FO	Formatting Objects
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
SGML	Standard Generalized Markup Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language (for) Transformations

1 Einleitung

1.1 Vorwort

Viele Unternehmen erstellen Dokumentationen und Anleitungen für ihre Produkte bis heute in mehr oder weniger unstrukturierter Form mit Hilfe von Textverarbeitungsprogrammen. Auch wenn diese Vorgehensweise in einigen Fällen zu durchaus präsentablen Resultaten führt, gibt es jedoch eine Vielzahl von Argumenten, die gegen deren Verwendung spricht.

Ist die Erzeugung eines PDF-Dokuments aus einem Textverarbeitungsprogramm meist noch problemlos möglich, müssen automatisch erzeugte HTML-Ausgaben, beispielsweise für Online-Handbücher, in nahezu jedem Fall nachbearbeitet werden.

Oftmals wird in diesen Bereichen zunächst eine entsprechende Version des Dokuments mit einem Textverarbeitungsprogramm erstellt und diese dann mehr oder weniger gut in fertig entwickelte HTML-Templates kopiert. Dass bei dieser Arbeitsweise zwangsläufig Reibungsverluste entstehen, liegt auf der Hand. Diese erhöhen sich proportional zur Anzahl der zu erzeugenden Zielformate und zur Anzahl der in den Entwicklungsprozess involvierten Mitarbeiter.

Spätestens bei der Aktualisierung einer solchen Sammlung von Dokumenten, sei es auch nur zur Korrektur eines zu spät entdeckten Rechtschreibfehlers, wird der enorme Verwaltungsaufwand und damit auch der nicht zu vernachlässigende ökonomische Aufwand ersichtlich.

Im Rahmen dieser Arbeit sollen Technologien erläutert werden, welche eben diesen Aufwand auf ein wirtschaftlich vertretbares Maß reduzieren und Redundanzen in der Datenhaltung weitestgehend vermeiden. Durch die Verwendung von mittlerweile den Kinderschuhen entwachsenen XML-Technologien muss im Idealfall nur ein einziges Quelldokument existieren, aus welchem – sprichwörtlich auf Knopfdruck – die gewünschten Zielformate erzeugt werden.

Gleichermaßen soll aber auch nicht verschwiegen werden, dass die Nutzung dieser Technologien in der Vorbereitungsphase ein gewisses Maß an Erfahrung und Fachwissen, einen höheren Arbeitsaufwand und eine klar strukturierte Arbeitsweise erfordern und nicht grundsätzlich für jede Dokumentation das Maß der Dinge darstellt.

1.2 Zielsetzung

Ziel dieser Arbeit soll neben der Erörterung theoretischer Grundlagen die Entwicklung eines praxistauglichen Benutzerhandbuchs für die Software „WPS-Manager 5.0“ der Schweißtechnischen Lehr- und Versuchsanstalt Halle GmbH sein. Dieses Benutzerhandbuch soll zum einen als Online-Hilfe in Form von mehreren, durchsuchbaren und untereinander verlinkten XHTML-Dokumenten, zum anderen als druckbares PDF-Dokument entstehen. Beide Zielformate sollen sprachlich und inhaltlich optimiert und optisch der Corporate Identity des Unternehmens angepasst werden.

1.3 Aufbau

Im ersten Teil dieser Arbeit werden zunächst die theoretischen Grundlagen des Cross-Media-Publishings, des XML-Formats im Allgemeinen und des DocBook-XML-Formats im Besonderen, sowie spezifische Grundlagen der gestalterischen und sprachlichen Umsetzung von Anleitungen erläutert. Zusätzlich wird aufgrund der vielfältigen Ausgabemöglichkeiten auf Endgeräten auch auf die Möglichkeiten des Responsive Webdesign eingegangen.

Im zweiten Teil dieser Arbeit werden die theoretischen Grundlagen anhand des zu erstellenden Benutzerhandbuchs an konkreten Beispielen angewandt und praxisnah umgesetzt.

2 Theoretische Grundlagen

2.1 Cross-Media-Publishing

Neben Cross-Media-Publishing beschreiben die Begriffe Single-Source-Publishing, Multi-Format-Publishing oder Multi-Channel-Publishing grundsätzlich den gleichen Sachverhalt, wenn auch teilweise aus anderen Sichtweisen, und können deshalb durchaus synonym verwendet werden.

Der Begriff des Cross-Media-Publishings beschreibt im Allgemeinen die mehrfache Verwendung von Informationen oder medialen Daten für verschiedene Ausgabemedien. Dabei liegt das Hauptaugenmerk auf einer medienneutralen, also nicht an ein spezifisches Ausgabemedium oder -format gebundenen, Erstellung und Pflege dieser Informationen (vgl. [INFOLOX]). Ziel des Cross-Media-Publishings ist die Publizierbarkeit der Informationen in möglichst vielen Ausgabemedien. Dies können beispielsweise klassische Printprodukte, wie Zeitschriften oder Bücher, oder moderne Medien, wie Veröffentlichungen im Internet (etwa als Artikel auf Websites), Kataloge auf CD-ROMs, E-Books oder Online-Hilfen sein. Auch Ausgabeformate, die nicht zwingend an ein einziges Medium gebunden sind, etwa PDF- oder PostScript-Dokumente, welche sowohl am Bildschirm betrachtet als auch gedruckt werden können, stellen mögliche Zielformate des Cross-Media-Publishings dar (vgl. [CONTENTMANAGER]).

Die optimale Grundlage zum Erreichen der Medienneutralität bietet aufgrund seiner Flexibilität das XML-Format, welches im nachfolgenden Abschnitt erläutert wird.

2.2 XML

XML, die Extensible Markup Language, ist eine Meta-Auszeichnungssprache – eine Sprache, die es ihrerseits ermöglicht, weitere Auszeichnungssprachen zu entwickeln. Der große Vorteil von XML ist, dass ihre Struktur und ihr Vokabular nicht fest vorgegeben ist, sondern vom Benut-

zer, unter Beachtung gewisser Grundregeln, frei definiert werden kann. XML ist daher in ihrem Sprachumfang nicht begrenzt, sondern kann nahezu beliebig erweitert werden. Nicht zuletzt der Fakt, dass ein XML-Dokument auch von einem Menschen lesbar ist, hat XML dazu verholfen, in vielen (Publishing-)Systemen zum kleinsten gemeinsamen Nenner für den Datenaustausch zu werden.

Die erste Spezifikation von XML wurde im Februar 1998 vom W3C in Form einer Empfehlung (Recommendation) herausgegeben und liegt aktuell seit November 2008 in der fünften Version vor (*[W3C 1]*).

Wie erwähnt, legte die Fähigkeit von XML, ihre Struktur frei definieren zu können, den Grundstein für die Entwicklung weiterer Auszeichnungssprachen, die ihrerseits auf XML basieren. Bekannte Vertreter dieser Sprachen sind beispielsweise XHTML, SVG, das OpenDocument-Austauschformat und, für diese Arbeit von besonderer Relevanz, DocBook und XSL-FO.

Auch wenn hier keine vollständige Abhandlung über die Entwicklung des XML-Formats und dessen Anwendungen gegeben werden soll, sei an dieser Stelle die Auszeichnungssprache HTML erwähnt, die wie XML selbst aus der Auszeichnungssprache SGML hervorging. Die nachstehende Abbildung zeigt den Ursprung der Auszeichnungssprachen HTML und XHTML aus SGML bzw. XML, sowie deren Zusammenhänge.

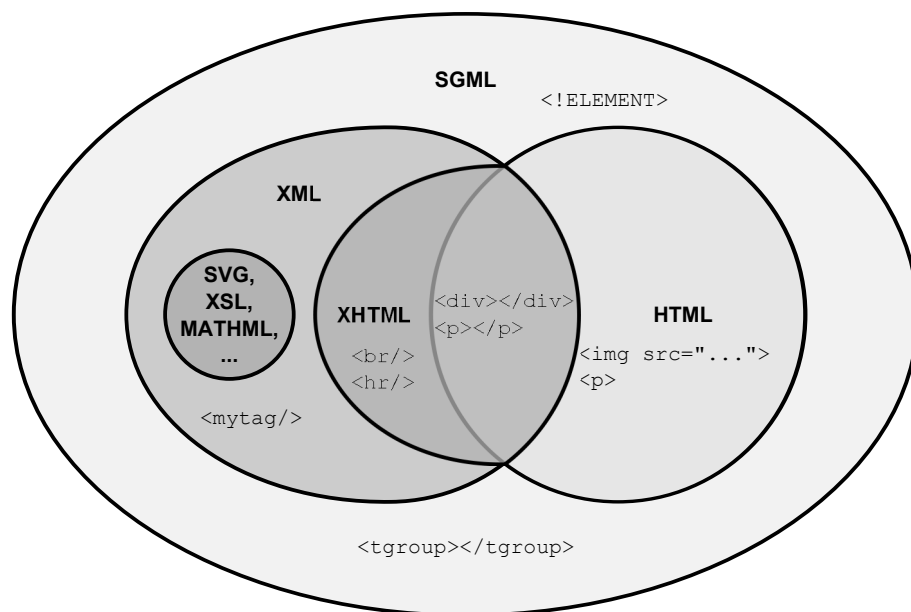


Abbildung 1: Zusammenhang der Meta-, Auszeichnungs- und Transformationssprachen (nach [USEGROUP])

Der direkte Vergleich von (X)HTML und XML an einem konkreten Beispiel soll deren Unterschied verdeutlichen. Wie bereits erwähnt, hat (X)HTML eine fest vorgegebene Struktur in Form bestimmter Elemente und Attribute, wogegen eben diese Elemente und Attribute in XML frei definiert werden können, wie das nachfolgende Beispiel zeigt (vgl. [TIDWELL 2002], S. 5):

```
01 <td>12304</td>
```

Listing 1: Tabellenzelle in (X)HTML

Das Listing zeigt das Element einer Tabellenzelle in (X)HTML. Der Ziffernfolge in diesem Element kann allerdings nicht ohne weiteres eine konkrete Bedeutung zugeordnet werden. So kann es sich dabei um eine Telefonnummer, eine Postleitzahl, ein Passwort oder einfach eine zufällige Ziffernfolge handeln. Für die reine Ausgabe in einem Browser ist die Bedeutung des Inhalts nicht relevant, lediglich die Information, dass es sich um eine Tabellenzelle innerhalb einer Tabelle handelt ist für den Browser wichtig. Relevant wird die Bedeutung des Elements dann, wenn die Daten auch für andere Zwecke, als zur reinen Darstellung im Browser genutzt werden sollen. Hier wäre es wesentlich sinnvoller, dem Element

einen Namen zu geben, mit welchem dessen Inhalt eine Bedeutung zugeordnet werden kann. An dieser Stelle kommt die freie Definierbarkeit von XML ins Spiel, so dass das Listing nun folgendermaßen aussehen könnte:

```
01  <postleitzahl>12304</postleitzahl>
```

Listing 2: Element in XML

Dem Element ist nun durch seinen Namen eine Bedeutung zuzuordnen. Somit besteht beispielsweise die Möglichkeit, alle Elemente mit dem Namen `postleitzahl` zu gruppieren, zu sortieren, als Liste darzustellen oder gar in einem komplett anderen Dateiformat, wie später noch gezeigt werden soll, auszugeben.

Ein weiterer Vorteil, der die Verwendung von XML mit sich bringt, ist die konsequente Trennung von Inhalt, Layout und Struktur.

2.2.1 Aufbau eines XML-Dokuments

Ein XML-Dokument besitzt eine in sich verschachtelte Struktur von Elementen und Attributen, die grundsätzlich frei definierbar ist. Soll jedoch eine bestimmte Struktur, also eine Art Grammatik der Sprache festgelegt werden, kann dies über eine DTD (Document Type Definition) oder andere Schemasprachen erfolgen. Nur so ist eine Überprüfung der in *Abschnitt 2.2.3* beschriebenen Validität oder Gültigkeit möglich.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <!DOCTYPE buch SYSTEM "buch.dtd">
03  <buch>
04      <buchtitel>Das Hallo Welt Buch</buchtitel>
05      <autor>Dipl.-Ing. (FH) Patrick Henkel</autor>
06      <kapitel>
07          <kapiteltitel>Erstes Kapitel</kapiteltitel>
08          <absatz>Diese Buch handelt von einer allseits
beliebten Aussage: Hallo Welt!</absatz>
09      </kapitel>
```

10 `</buch>`

Listing 3: Aufbau eines XML-Dokuments

- 01 XML-Deklaration; Sie zeigt an, dass es sich beim vorliegenden Dokument um ein XML-Dokument handelt. Der Wert des Attributs `version` legt fest, dass das Dokument der XML-Spezifikation der Version 1.0 entspricht. Der Wert des optionalen Attributs `encoding` legt die Zeichencodierung des Dokuments fest. Wird das Attribut nicht angegeben, dann wird vom Interpreter automatisch die Zeichencodierung UTF-8 verwendet.
- 02 Dokumenttyp-Deklaration; Sie ist ebenfalls optional und legt das Regelwerk, hier eine DTD, fest, nach dem das Dokument aufgebaut ist. Nur mit einem Regelwerk kann ein XML-Dokument auf seine Gültigkeit geprüft werden.
- 03 Start-Tag des Wurzelements; Ein wohlgeformtes XML-Dokument enthält genau ein Wurzelement. Ihm sind alle anderen Elemente untergeordnet.
- 10 End-Tag des Wurzelements

2.2.2 Elemente, Tags und Attribute

Die Inhalte eines XML-Dokuments werden durch Elemente und Attribute strukturell definiert. Dabei besteht jedes Element aus einem Start-Tag, einem End-Tag und dem dazwischenliegenden Elementinhalt. Zusätzlich kann das Start-Tag ein oder mehrere Attribute enthalten, die beispielsweise das Element näher beschreiben. Neben den Elementen mit Start- und End-Tag gibt es die leeren Elemente, bei denen Start- und End-Tag sowie optionale Attribute in einem Tag kombiniert sind. Die Namen der Elemente und Attribute dürfen nur mit Groß- oder Kleinbuchstaben oder einem Unterstrich beginnen und dürfen nur alphanumerische Zeichen, sowie Binde- und Unterstrich enthalten. Wichtig ist zudem, dass der Name im Start- und End-Tag absolut identisch ist. Die nachfolgende Abbildung des Aufbaus eines Elements soll diese Zusammenhänge nochmals verdeutlichen.

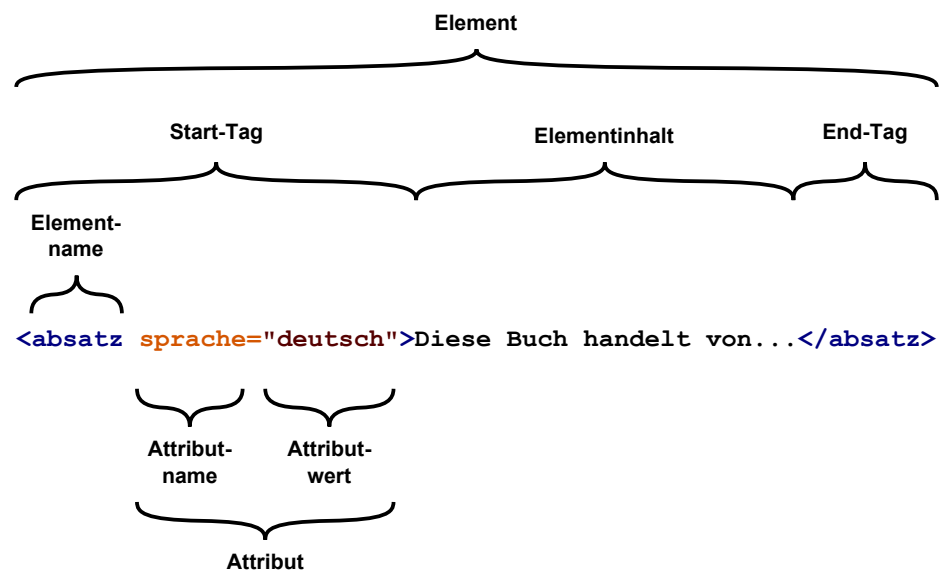


Abbildung 2: Aufbau eines Elements

Neben Elementen mit Start- und End-Tags können in einem XML-Dokument auch so genannte leere Elemente vorkommen. Diese Elemente können ebenfalls über Attribute verfügen. Leere Elemente haben dagegen keinen Elementinhalt und (in der Regel) kein End-Tag, sondern einen Schrägstrich vor der schließenden spitzen Klammer (vgl. [W3C 2]).

2.2.3 Wohlgeformtheit und Gültigkeit

Jedes XML-Dokument, welches den Regeln der XML-Spezifikation entspricht, gilt als wohlgeformt. Insbesondere die im vorherigen Abschnitt beschriebenen Regeln, sind Voraussetzungen für wohlgeformte XML-Dokumente. Wird also beispielsweise ein Element mit einem Start-Tag nicht mit einem gleichlautendem End-Tag beendet oder der Elementname beginnt mit einer Ziffer, so ist das Dokument nicht wohlgeformt. Gleiches gilt, wenn mehr als ein Wurzelement existiert. Grundsätzlich dürfen XML-Parser nicht wohlgeformte XML-Dokumente weder anzeigen noch versuchen zu korrigieren. Er darf lediglich eine Fehlermeldung ausgeben und muss dann das Parsen des Dokuments abbrechen (vgl. [HAROLD 2004], S. 164).

Wird im XML-Dokument eine Dokumenttyp-Deklaration angegeben, beispielsweise in Form einer DTD oder XML-Schema, kann das XML-Doku-

ment gegen diese validiert werden. Es wird damit geprüft, ob die Struktur des Dokuments den dort definierten Regeln entspricht. Entspricht das XML-Dokument diesen Regeln, so ist das XML-Dokument valide bzw. gültig.

2.3 Namensräume

Man stelle sich folgendes Problem vor: in einem großen Unternehmen existieren zwei Mitarbeiter mit dem Namen Peter Müller. In diesem Fall sind Verwechslungen vorprogrammiert. Dieses Problem kann jedoch vermieden werden, wenn man nicht nur von Peter Müller spricht, sondern von Peter Müller aus der Buchhaltung und von Peter Müller aus der Lagerverwaltung. In diesem Beispiel entsprechen Buchhaltung und Lagerverwaltung dem Namensraum.

Genau dieses Prinzip wird auch bei XML-Dokumenten verwendet.

Die Verwendung von Namensräumen oder auch Namespaces dient also dazu, Namenskonflikte zu vermeiden, welche auftreten, wenn mehrere Elemente innerhalb eines XML-Dokuments den gleichen Namen haben und bietet damit die Möglichkeit, mehrere XML-Sprachen nebeneinander zu verwenden.

Die Angabe der Namensräume erfolgt innerhalb des Start-Tags der Elemente. Dazu wird in einem `xmlns`-Attribut eine URI angegeben, die meist auf eine Website verweist. Grund dieser Art der Angabe ist, dass URIs einzigartig sind, also nicht mehrfach vorkommen können. Obwohl diese Adressen durchaus auf Dokumente verweisen können, müssen sie nicht wirklich existieren. Ein Beispiel für die Angabe des Namensraums zeigt nachstehendes Listing.

```
01 <html xmlns="http://www.w3.org/1999/xhtml">
02     ...
03 </html>
```

Listing 4: Angabe des Namensraums

Alle Kindelemente des `<html>`-Elements existieren demnach im angegebenen Namensraum. Im obigen Listing wird ein default-Namensraum angegeben. Sollen nun bestimmte Kindelemente einen anderen Namensraum verwenden, muss für diese Elemente ein neuer Namensraum angegeben werden. Dies kann entweder auf die gleiche Weise, wie im obigen Listing geschehen oder unter Verwendung von Präfixen. Der Präfix repräsentiert dann den an ihn gebundenen Namensraum. Ein Beispiel dafür zeigt nachstehendes Listing.

```
01  <html
02      xmlns="http://www.w3.org/1999/xhtml"
03      xmlns:m="http://www.w3.org/1998/Math/MathML"
04  >
05      ...
06      <m:math>
07          ...
08      </m:math>
09  </html>
```

Listing 5: Angabe eines zusätzlichen Namensraum mit einem Präfix

2.4 DocBook-XML

Wie bereits mehrfach erwähnt, liegt der große Vorteil der Verwendung des XML-Formats zur medienneutralen Speicherung von Daten in der nahezu freien Definierbarkeit dieses Formats. So wäre es ohne weiteres möglich, Definitionen für ein eigenes Format zu entwickeln oder ein vorhandenes Format durch Anpassung dessen Definitionen, der Document Type Definition oder kurz DTD, zu erzeugen. Allerdings muss zur medienneutralen Speicherung der Inhalte eines Benutzerhandbuchs oder einer technischen Dokumentation im Allgemeinen bzw. eines Benutzerhandbuchs im Speziellen kein neues Format entwickelt werden. Es existiert bereits ein speziell für diese Zwecke entwickeltes Format – das DocBook-XML-Format.

Das DocBook-Format ist ebenfalls eine Auszeichnungssprache, dessen Entwicklung seit 1991 stattfindet. Sie wurde ursprünglich von den Fir-

men O'Reilly & Associates und HaL Computer Systems entwickelt und beinhaltete damals bereits viele Einflüsse aus SGML-basierten Datenaustausch-Projekten. Seit 1998 wird die Weiterentwicklung von DocBook durch die Organisation OASIS (Organization for the Advancement of Structured Information Standards) vorangetrieben, die unter anderem auch für die Entwicklung des OpenDocument-Formats verantwortlich ist. Durch deren Federführung wurden die bis dahin nur in SGML spezifizierten Definitionen des Formats auch in XML zur Verfügung gestellt – nicht zuletzt durch die Entwicklung von XML selbst angetrieben –, so dass DocBook seit der Version 4.0 sowohl als SGML- als auch XML-DTD existiert. Ab DocBook 5.0 basiert das Regelwerk nur noch auf einer XML-DTD bzw. der Schemasprache RELAX NG, wobei letztere favorisiert wird.

Obwohl das DocBook-Format ursprünglich für technische Dokumentationen entwickelt wurde, bietet es genügend Freiheiten für andere Publikationen. So können durchaus ganze Bücher und Buchreihen damit entwickelt werden.

Das nachfolgende Listing zeigt exemplarisch den Grundaufbau eines einfachen DocBook 5.0-XML-Dokuments. Es entspricht inhaltlich dem Beispiel aus *Listing 3* in *Abschnitt 2.2.1*, verweist aber auf eine andere DTD und besitzt DocBook 5.0-spezifische Elemente.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML
    5.0b3//EN" "http://www.oasis-open.org/docbook/
    xml/5.0b3/dtd/docbook.dtd">
03  <book xmlns="http://docbook.org/ns/docbook">
04      <info>
05          <title>Das Hallo Welt Buch</title>
06          <author>
07              <personname>
08                  <honorific>Dipl.-Ing. (FH)</honorific>
09                  <firstname>Patrick</firstname>
10                  <surname>Henkel</surname>
11              </personname>
12          </author>
13      </info>
```

```
14     <chapter>
15         <title>Erstes Kapitel</title>
16         <para>Diese Buch handelt von einer allseits
beliebten Aussage: Hallo Welt!</para>
17     </chapter>
18 </book>
```

Listing 6: einfaches DocBook 5.0-XML-Dokument

- 01 Das Dokument beginnt als XML-Dokument mit der XML-Deklarationszeile.
- 02 In der Dokumenttyp-Deklaration wird dem Wurzelement `<book>` die entsprechende DocBook 5.0-DTD zugewiesen.
- 03 Das Wurzelement des Dokuments ist das `<book>`-Element.
- 04 Im `<info>`-Element werden an dieser Stelle die Metadaten des Buchs, also Buchtitel, Angaben zum Autor, Veröffentlichungsdatum usw. notiert. Auch Kapitel, Artikel, Abschnitte usw. können in DocBook 5.0 dieses Element enthalten.
- 14 Die einzelnen Teile eines Buchs können unter anderem mit `<chapter>`- (Kapitel) und `<section>`-Elementen (Abschnitte) festgelegt werden. Weitere Möglichkeiten wären etwa `<part>` (Teil), `<article>` (Artikel) und `<sect1>` bis `<sect5>` (verschachtelte Abschnitte).

Neben den erwähnten Elementen existieren innerhalb der DocBook-Spezifikation eine Vielzahl weiterer Elemente zur Strukturierung des Inhalts. Erwähnenswert sind davon insbesondere:

- Listen (z.B. ungeordnete Listen, geordnete Listen, Anweisungslisten zur Beschreibung von Vorgängen und Frage-und-Antwort-Listen)
- Tabellen (u.a. mit Kopf- und Fußzeilen)
- Querverweise (Verweise auf Ziele im Text, indirekte Verweise auf z.B. andere Kapitel, Verweise auf externe URLs)
- Elemente zum Auszeichnen von Elementen einer grafischen Benutzeroberfläche

- Elemente zum Einbetten von Multimediaobjekten (Abbildungen, Screenshots)

Die Transformationsanweisungen, um Dokumente im DocBook-XML-Format in die gewünschten Zielformate umzuwandeln, werden in entsprechenden XSL-Stylesheets definiert. Grundsätzlich bieten sich dafür zwei Vorgehensweisen an. Zum einen existieren neben der DocBook-DTD bzw. dem Schema schon eine Reihe vorgefertigter XSL-Stylesheets, die unter anderem Transformationen in XSL-FO-, (X)HTML- und HTMLhelp-Dokumente ermöglichen (siehe [DOCBOOK]). Durch Parameter können diese Stylesheets in gewissen Grenzen mittels der so genannten *customization layers* – XSL-Stylesheets, die die Standardwerte der Parameter in den ursprünglichen XSL-Stylesheets überschreiben, den eigenen Bedürfnissen angepasst werden und ermöglichen damit eine mehr oder weniger individuelle Gestaltung der Ausgabedokumente. In vielen Fällen, so auch bei der Entwicklung des im Rahmen dieser Arbeit entstandenen Benutzerhandbuchs, kann es jedoch von Vorteil sein, eigene XSL-Stylesheets zu entwickeln. Zwar kann im Allgemeinen der Aussage

You don't need a customization layer to customize how your HTML output looks. You can control your HTML presentation by simply adding your own CSS stylesheet to the standard DocBook HTML output. ([STAYTON 2003], S. 89)

von Bob Stayton zugestimmt werden, jedoch muss im Einzelfall die HTML-Struktur eigenen Bedürfnissen komplett angepasst und nicht nur die ausgegebene Struktur per CSS umformatiert werden.

2.5 XSL

Nachdem sich XML nach ihrer Einführung mehr und mehr zum Standard für den Austausch strukturierter Daten entwickelt hat und als solche immer mehr an Verbreitung gewann, bedurfte es einer flexiblen, relativ einfachen aber dennoch leistungsfähigen Sprache, um diese strukturierten Daten zu verarbeiten. Eben diese Sprache ist XSL, die Extensible Stylesheet Language – eine vom W3C standardisierte Sprache zur Formatierung und Verarbeitung von XML-Dokumenten.

Bei XSL handelt es sich jedoch weniger um eine Sprache ansich, als vielmehr um eine Sprachfamilie, die ihrerseits aus drei Teilen besteht.

XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts [...] ([W3C 3])

Diese drei Teile sind XSLT, XPath und XSL-FO und werden in den nachfolgenden Abschnitten erläutert.

2.6 XSLT

XSLT, die Extensible Stylesheet Language (for) Transformations, ist eine deklarativ-funktionale Sprache zur Transformation der Struktur eines XML-Dokuments in eine andere Struktur. In Templates – in sich geschlossene funktionale Einheiten – wird deklariert, was mit bestimmten Elementen geschehen soll. Dokumente, in denen XSLT verwendet wird, die so genannten XSL-Stylesheets, sind selbst valide XML-Dokumente, unterliegen also den bereits erläuterten Regeln für XML-Dokumente. Die Versionen XSLT 1.0 ([W3C 4]) und XSLT 1.1 ([W3C 5]) wurden vom britischen Informatiker James Clark entwickelt. Die Weiterentwicklung der Sprache, die aktuell in der Version XSLT 2.0 vorliegt, wird vom britischen Programmierer Michael Kay verantwortet. Seit Januar 2007 wird XSLT 2.0 vom W3C in Form einer Recommendation empfohlen ([W3C 6]). Seit Dezember 2010 existiert ein Working Draft für die nächste Generation XSLT 3.0 ([W3C 7]).

XSLT bietet unter anderem die Möglichkeit, XML-Dokumente neu zu strukturieren, in andere Formate umzuwandeln, diese in mehreren Ergebnisdokumenten auszugeben oder mehrere XML-Dokumente miteinander zu kombinieren.

Die dazu benötigten Formatierungsanweisungen werden in den XSL-Stylesheets definiert und gemeinsam mit den XML-Quelldokumenten durch einen XSLT-Prozessor verarbeitet oder besser gesagt: transformiert. Die nachstehende Abbildung zeigt das grundlegende Prinzip einer solchen Transformation.

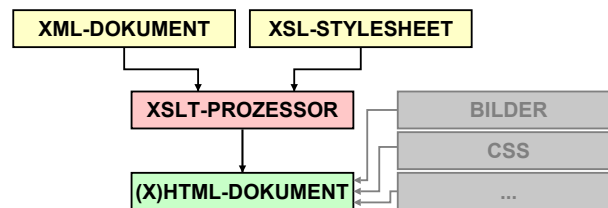


Abbildung 3: Prinzip der Erzeugung eines (X)HTML-Dokuments

Das folgende Listing zeigt ein noch recht übersichtliches XSL-Stylesheet, welches das DocBook-XML-Dokument in *Listing 6* aus *Abschnitt 2.4* in ein valides XHTML-Dokument transformiert.

```

01  <?xml version="1.0" encoding="UTF-8"?>
02  <xsl:stylesheet
03      xmlns="http://www.w3.org/1999/xhtml"
04      xmlns:xsl="http://www.w3.org/1999/XSL/Trans-
form"
05      xpath-default-namespace="http://docbook.org/ns/
docbook"
06      version="2.0">
07
08      <xsl:output
09          method="xhtml"
10          encoding="UTF-8"
11          doctype-public="-//W3C//DTD XHTML 1.1//EN"
12          doctype-system="http://www.w3.org/TR/xhtml11/
DTD/xhtml11.dtd"/>
13
14      <xsl:template match="/">
15          <html>
16              <head>
17                  <title><xsl:value-of select="book/info/
title"/></title>
18              </head>
19              <body>
20                  <xsl:apply-templates/>
21              </body>
22          </html>
23      </xsl:template>
  
```

```
24
25     <xsl:template match="book/info/title">
26         <h1><xsl:value-of select="."/></h1>
27     </xsl:template>
28
29     <xsl:template match="author">
30         <h2>
31             <xsl:text>von </xsl:text>
32             <xsl:value-of select="personname/honori-
33 fic"/>
34             <xsl:text> </xsl:text>
35             <xsl:value-of select="personname/firstna-
36 me"/>
37             <xsl:text> </xsl:text>
38             <xsl:value-of select="personname/surname"/>
39         </h2>
40     </xsl:template>
41
42     <xsl:template match="chapter">
43         <div class="kapitel">
44             <xsl:apply-templates/>
45         </div>
46     </xsl:template>
47
48     <xsl:template match="chapter/title">
49         <h3><xsl:value-of select="."/></h3>
50     </xsl:template>
51
52     <xsl:template match="para">
53         <p><xsl:apply-templates/></p>
54     </xsl:template>
55
56 </xsl:stylesheet>
```

Listing 7: XSL-Stylesheet

Nach der Transformation des XML-Quelldokuments sieht das Ergebnisdokument wie folgt aus:

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <!DOCTYPE html
03      PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.
      w3.org/TR/xhtml11/DTD/xhtml11.dtd">
04  <html>
05      <head>
06          <title>Das Hallo Welt Buch</title>
07      </head>
08      <body>
09          <h1>Das Hallo Welt Buch</h1>
10          <h2>von Dipl.-Ing. (FH) Patrick Henkel</h2>
11          <div class="kapitel">
12              <h3>Erstes Kapitel</h3>
13              <p>Diese Buch handelt von einer allseits be-
                  liebten Aussage: Hallo Welt!</p>
14          </div>
15      </body>
16  </html>
```

Listing 8: Quelltext des Ergebnisdokuments im XHTML-Format

Die nachstehende Abbildung zeigt das XHTML-Dokument im Browser.

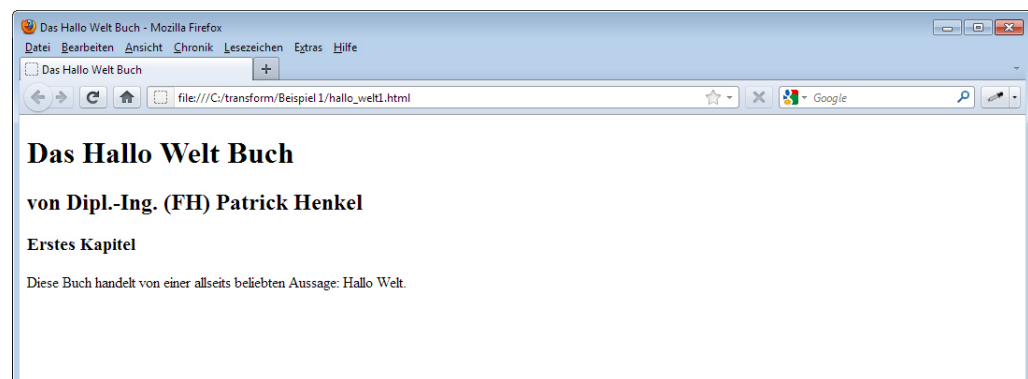


Abbildung 4: Ausgabe des Ergebnisdokuments im Firefox Browser

Dieses, zugegeben nicht sonderlich spektakuläre Beispiel, zeigt kaum die Stärken der Sprache XSL, demonstriert jedoch das grundlegende Prinzip der Transformation. Es zeigt deutlich, dass die so genannten Templates das zentrale Element von XSL-Stylesheets sind. Beim Trans-

formationsprozess werden die im `match`-Attribut angegebenen Elemente des XML-Dokuments entsprechend den Regeln des jeweiligen Templates formatiert.

Erwähnenswert ist an dieser Stelle die Verwendung des `xpath-default-namespace`-Attributs im Wurzelement von *Listing 7*. Da der default-Namensraum des Ergebnisdokuments auf den XHTML-Namensraum verweisen, innerhalb des XSL-Stylesheets jedoch auch der DocBook-Namensraum verwendet wird, würden Konflikte entstehen. Um diesen Konflikt zu vermeiden, kann entweder ein Präfix, wie in *Abschnitt 2.3* beschrieben, verwendet werden, welches im Wurzelement dem DocBook-Namensraum zugewiesen und den DocBook-Elementen innerhalb der XPath-Ausdrücke vorangestellt wird oder eben das in *Listing 7* dargestellte `xpath-default-namespace`-Attribut verwendet werden.

Neben der Möglichkeit, die Transformation mit einem externen XSLT-Prozessor, wie beispielsweise dem Saxon XSLT-Prozessor, zu transformieren, können diese Transformationen auch direkt in modernen Browsern durchgeführt werden. Diese besitzen einen integrierten XSLT-Prozessor. Firefox benutzt zur Transformation beispielsweise den Transformixx XSLT-Prozessor. Dieser unterstützt jedoch nur XSLT und XPath in den veralteten Versionen 1.0 und ist daher nur eingeschränkt nutzbar.

Um die Transformation im Browser durchzuführen, muss im XML-Dokument in Form einer so genannten Processing Instruction, oft auch als PI abgekürzt, die Information des zu verwendenden XSL-Stylesheets angegeben werden. Diese Processing Instructions werden im XML-Quelldokument direkt nach der XML-Deklaration wie folgt angegeben:

```
01  <?xml-stylesheet type="text/xsl"
    href="stylesheet.xsl"?>
```

Listing 9: Angabe des Stylesheets in einer Processing Instruction im XML-Quelldokument

Das `type`-Attribut gibt dabei den Typ des Stylesheets an (hier den Typ `text/xsl`), im `href`-Attribut wird der Pfad zum XSL-Stylesheet angegeben.

Neben den XSL-Stylesheets werden oftmals die Cascading Stylesheets (CSS) als einfacher zu verstehende Alternative angesehen. In gewissen, sehr engen Grenzen, kann dem zugestimmt werden. Allerdings muss dabei beachtet werden, dass Cascading Stylesheets zur reinen Formatierung von bestehenden Strukturen entwickelt wurden. So kann mit CSS das Aussehen der Elemente eines XML-basierten Dokuments festgelegt, jedoch nicht deren Reihenfolge und Struktur verändert oder gar neue Elemente erzeugt werden, wie es mit XSLT möglich ist (vgl. [HERPERS 2002], S. 23). Beide Sprachen sind letztenendes für verschiedene Zwecke entwickelt worden. Es ist daher wenig zielführend, sie als Konkurrenten gegenüber zu stellen. Sie sollten eher für die jeweiligen Zwecke sinnvoll eingesetzt werden.

Wie erwähnt, stellt XSLT nur einen Teil der Extensible Stylesheet Language dar. Zur XSL-Familie gehören daneben noch XPath und XSL-FO, die in den nachfolgenden Abschnitten beschrieben werden.

2.7 XPath

XPath ist eine vom W3C entwickelte und 1999 in der Version 1.0 verabschiedete Abfragesprache für XML-Dokumente ([W3C 8]). Seit Januar 2007 liegt XPath in der Version 2.0 vor ([W3C 9]). Diese Sprache ermöglicht es, einzelne Teile eines XML-Dokuments zu adressieren. Es ist also mit XPath möglich, bestimmte Elemente innerhalb der Baumstruktur eines XML-Dokuments anzusprechen. Dabei ist XPath nicht allein auf die Verwendung innerhalb von XSL-Stylesheets beschränkt, sondern findet beispielsweise auch in Skriptsprachen wie PHP Anwendung.

2.7.1 Knotentypen

Die Baumstruktur eines XML-Dokuments besteht aus verschiedenen Arten von Knoten, die im übertragenen Sinn die Äste des Baumes repräsentieren. XPath unterscheidet dabei zwischen sieben verschiedenen Knotentypen (vgl. [TIDWELL 2002], S. 44):

- Wurzelknoten

- Elementknoten
- Attributknoten
- Textknoten
- Kommentarknoten
- Verarbeitungsanweisungsknoten
- Namensraumknoten
- Lokalisierungspfade

Die so genannten Lokalisierungspfade mit deren einzelnen, durch einen Schrägstrich voneinander getrennten, Lokalisierungsschritten stellen die wohl häufigste Anwendungsart von XPath dar. In ihrer Schreibweise sind sie der Schreibweise von Pfadangaben in einem Verzeichnissystem sehr ähnlich. Mit ihnen können gezielt Knoten adressiert werden, wie die nachfolgenden Beispiele von XPath-Ausdrücken in einem XSL-Stylesheet verdeutlichen.

```
01 <xsl:value-of select="."/>
```

Der XPath-Ausdruck liefert den Inhalt des aktuellen Kontextknotens.

```
01 <xsl:template match="section/para">...</xsl:template>
```

Der XPath-Ausdruck wendet das Template auf alle `<para>`-Elemente an, deren Elternelement das `<section>`-Element ist.

```
01 <xsl:apply-templates match="para"/>
```

Der XPath-Ausdruck wählt alle untergeordneten `<para>`-Elemente des aktuellen Kontextknotens aus.

Wichtig bei der Verwendung von XPath-Ausdrücken ist die Unterscheidung zwischen absoluten und relativen Ausdrücken. Genau wie bei Pfadangaben in Dateisystemen ist bei relativen Ausdrücken die aktuelle Position, bei XPath-Ausdrücken eben der aktuelle Kontextknoten, zu beachten, da die Ausdrücke relativ zu diesem angegeben werden. Sollen bestimmte Knoten dagegen absolut angesprochen werden, wird im entsprechenden XPath-Ausdruck ein Schrägstrich vorangestellt, der den

XSLT-Prozessor dazu veranlasst, den entsprechenden Knoten, ausgehend vom Wurzelknoten des Dokuments, anzusprechen.

2.7.2 Prädikate

Wesentlich komplexere Pfadangaben können in XPath mit so genannten Prädikaten realisiert werden. Diese Prädikate funktionieren wie Filter, die die Auswahl der Knoten, welche durch den reinen XPath-Ausdruck ermittelt wurden, noch weiter einschränken können.

Jedes Prädikat wird ausgewertet, und in einen Booleschen Wert konvertiert (entweder true oder false). Ist das Prädikat für einen gegebenen Knoten true, wird dieser Knoten ausgewählt; ansonsten wird der Knoten nicht ausgewählt. ([TIDWELL 2002], S. 56)

Die nachfolgenden Beispiele stellen die Verwendung von XPath-Ausdrücken mit Prädikaten dar. Aus didaktischen Gründen werden die Ausdrücke in ihrer ausführlichen Schreibweise notiert. Daneben existieren für viele Ausdrücke auch Kurzschreibweisen, wie in *Abschnitt 2.7.4* beschrieben wird.

```
01 <xsl:value-of select="para[position() =  
last()]"/>
```

Der XPath-Ausdruck liefert den Inhalt des letzten `<para>`-Elements des aktuellen Kontextknotens.

```
01 <xsl:value-of select="para[position() = 4]"/>
```

Der XPath-Ausdruck liefert den Inhalt des vierten `<para>`-Elements des aktuellen Kontextknotens.

Zuletzt soll im nachfolgenden Beispiel die Verwendung eines Prädikats zur Filterung nach einem bestimmten Attribut demonstriert werden.

```
01 <xsl:value-of select="para[attribute::deutsch]"/>
```

Der XPath-Ausdruck liefert in XSLT 2.0 den Inhalt aller `<para>`-Elemente des aktuellen Kontextknotens, welche über das Attribut `deutsch` verfü-

gen. In XSLT 1.0 wird lediglich der Inhalt des ersten vom Kontextknoten aus gesehenen `<para>`-Elements, welches über das Attribut `deutsch` verfügt, zurückgegeben.

2.7.3 Achsen

Um in XPath-Ausdrücken die Beziehungen zwischen den einzelnen Elementknoten innerhalb der Baumstruktur abbilden zu können, stellt XPath eine Reihe von so genannten Achsen zur Verfügung. Auch wenn die Arbeit mit den Achsenbezeichnern zunächst gewöhnungsbedürftig erscheint, so stellt sie doch einen elementaren Bestandteil der XPath-Notation dar. Die Bezeichnungen der einzelnen Achsen sind zwar grundsätzlich selbsterklärend, sollen im Folgenden jedoch noch im Detail erläutert werden.

Die Kreise in den nachstehenden Abbildungen stellen die einzelnen Elementknoten innerhalb der Baumstruktur dar. Der fett umrandete Kreis repräsentiert den aktuellen Kontextknoten, aus dessen Sicht die jeweilige Achse betrachtet wird. Die grau gefüllten Kreise entsprechen den zur jeweiligen Achse gehörenden Knoten (vgl. [TIDWELL 2002], S. 54-56 und [SELFHTML]).

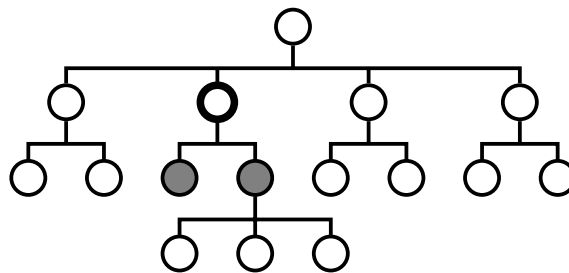


Abbildung 5: *child-Achse*

Ein XPath-Ausdruck, der die *child-Achse* (Kind-Achse) anspricht, liefert alle Kindelemente des aktuellen Kontextknotens.

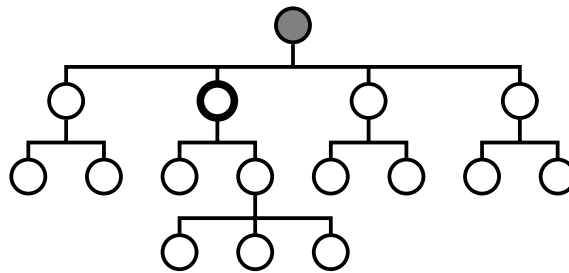


Abbildung 6: parent-Achse

Ein XPath-Ausdruck, der die parent-Achse (Eltern-Achse) anspricht, liefert das Elternelement des aktuellen Kontextknotens.

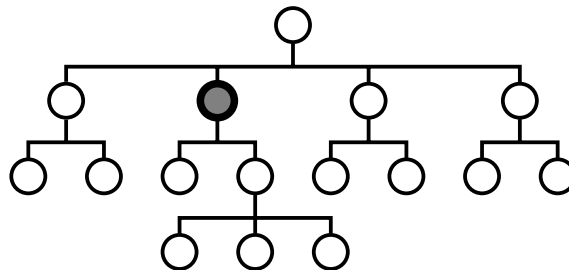


Abbildung 7: self-Achse

Ein XPath-Ausdruck, der die self-Achse (Selbst-Achse) anspricht, liefert den aktuellen Kontextknoten selbst.

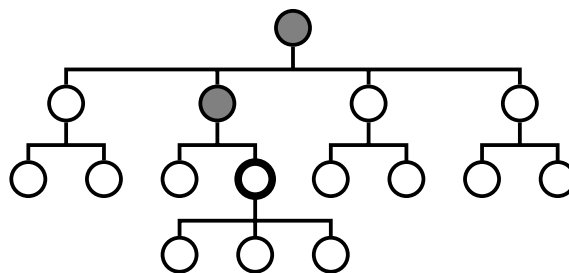


Abbildung 8: ancestor-Achse

Ein XPath-Ausdruck, der die ancestor-Achse (Vorfahren-Achse) anspricht, liefert alle Elemente, die Vorfahren, also Eltern, Großeltern usw. des aktuellen Kontextknotens sind.

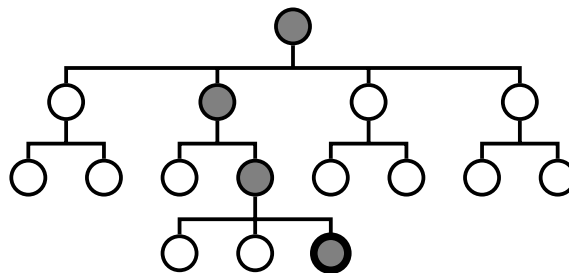


Abbildung 9: *ancestor-or-self-Achse*

Ein XPath-Ausdruck, der die ancestor-or-self-Achse (Vorfahren-oder-Selbst-Achse) anspricht, liefert alle Elemente, die Vorfahren, also Eltern, Großeltern usw. des aktuellen Kontextknotens sind, sowie den aktuellen Kontextknoten selbst.

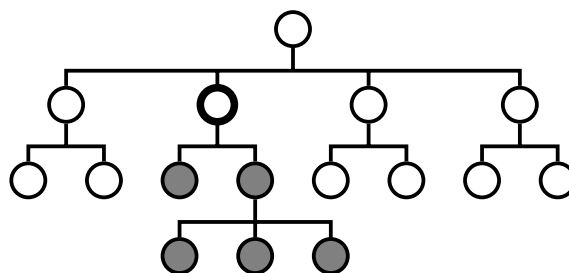


Abbildung 10: *descendant-Achse*

Ein XPath-Ausdruck, der die descendant-Achse (Nachfahren-Achse) anspricht, liefert alle Elemente, die Nachfahren, also Kinder, Kindeskindern usw. des aktuellen Kontextknotens sind.

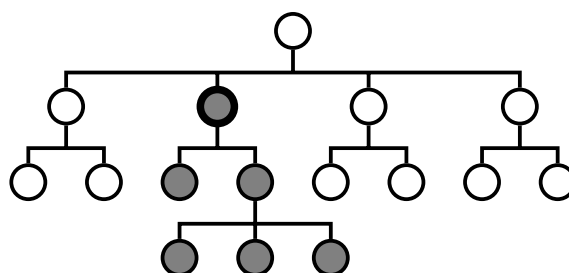


Abbildung 11: *descendant-or-self-Achse*

Ein XPath-Ausdruck, der die descendant-or-self-Achse (Nachfahren-oder-Selbst-Achse) anspricht, liefert alle Elemente, die Nachfahren, also

Kinder, Kindeskindern usw. des aktuellen Kontextknotens sind, sowie den aktuellen Kontextknoten selbst.

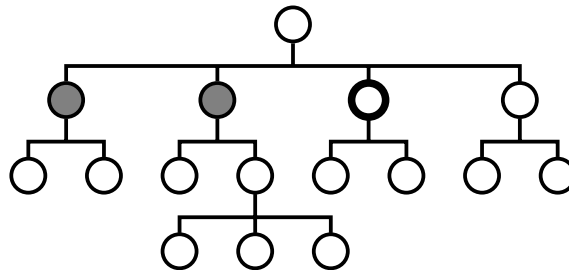


Abbildung 12: *preceding-sibling-Achse*

Ein XPath-Ausdruck, der die preceding-sibling-Achse (vorhergehende Geschwister-Achse) anspricht, liefert alle Elemente, die vorhergehende Geschwister des aktuellen Kontextknotens sind, also auf der gleichen Hierarchieebene wie dieser stehen.

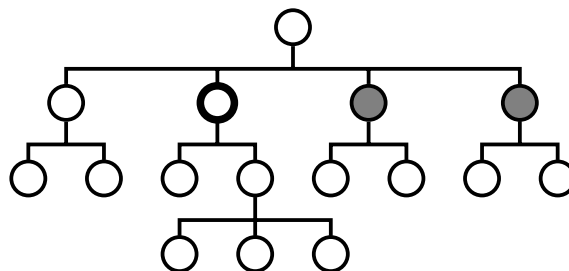


Abbildung 13: *following-sibling-Achse*

Ein XPath-Ausdruck, der die following-sibling-Achse (nachfolgende Geschwister-Achse) anspricht, liefert alle Elemente, die nachfolgende Geschwister des aktuellen Kontextknotens sind, also auf der gleichen Hierarchieebene wie dieser stehen.

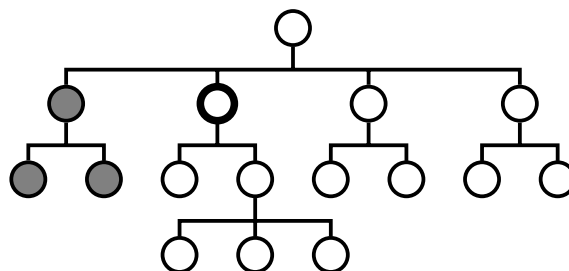


Abbildung 14: *preceding-Achse*

Ein XPath-Ausdruck, der die preceding-Achse (Vorgänger-Achse) anspricht, liefert alle Elemente, die vor dem aktuellen Kontextknoten in der Baumstruktur auftreten. Die Hierarchieebene dieser Elementknoten ist dabei egal.

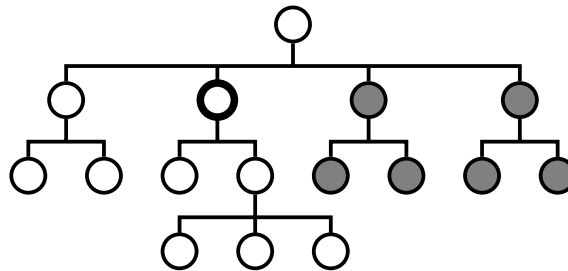


Abbildung 15: following-Achse

Ein XPath-Ausdruck, der die following-Achse (Nachfolger-Achse) anspricht, liefert alle Elemente, die nach dem aktuellen Kontextknoten in der Baumstruktur auftreten. Die Hierarchieebene dieser Elementknoten ist dabei egal.

Das folgende Beispiel soll nochmals die Verwendung der Achsen innerhalb eines XPath-Ausdrucks verdeutlichen. Als XML-Dokument dient *Listing 6*. Der aktuelle Kontextknoten sei das Element `<para>`.

```
01 <xsl:value-of select="preceding::title[1]" />
```

Das Ergebnis des XPath-Ausdrucks liefert den Inhalt des dem `<para>`-Element vorhergehenden Elements `<title>`, also den String „Erstes Kapitel“. Ohne das Präfix `[1]` würde der XPath-Ausdruck die Inhalte aller vorhergehenden `<title>`-Elemente zurückliefern. Mit dem Präfix wird dagegen nur der Inhalt des – aus Sicht des aktuellen Kontextknotens – ersten passenden Elements zurückgeliefert.

2.7.4 Verkürzte Schreibweisen

Neben den expliziten Angaben der Achsen existieren für einige der Achsenbezeichnungen und Angaben von Prädikaten verkürzte Schreibweisen. So ist der Ausdruck

```
01 <xsl:value-of select="child::abschnitt/
    child::absatz[position() = 1]"/>
```

gleichbedeutend mit dem Ausdruck

```
01 <xsl:value-of select="abschnitt/absatz[1]"/>
```

und verweist in beiden Schreibweisen auf das erste `<absatz>`-Kindelement des Elements `<abschnitt>`.

Die Achsenbezeichner der `child`-Achse können bei der verkürzten Schreibweise einfach weggelassen werden. Gleiches gilt für die explizite Angabe der XPath-Funktion `position()`. Diese verkürzte Prädikatschreibweise ähnelt der Angabe eines Elements in einem indexierten Array, wobei hier im Gegensatz zu Arrays das erste Element nicht den Index 0 sondern tatsächlich den Index 1 hat.

Einige häufig verwendete verkürzte Schreibweisen sind in der nachstehenden Tabelle aufgelistet.

ausführlich	verkürzt	Erläuterung
<code>self::para</code>	<code>./para</code>	Wählt den aktuellen Kontextknoten, wenn er vom Typ <code>para</code> ist
<code>parent::section</code>	<code>../section</code>	Wählt das Elternelement des aktuellen Kontextknotens, wenn es vom Typ <code>section</code> ist
<code>/descendant-or-self::node()/absatz</code>	<code>//absatz</code>	Wählt alle Elementknoten vom Typ <code>absatz</code> absolut vom Wurzelement
<code>para[attribute::deutsch]</code>	<code>para[@deutsch]</code>	Wählt alle Elementknoten vom Typ <code>para</code> aus, die das Attribut <code>deutsch</code> besitzen

Tabelle 1: Verkürzte Schreibweisen in XPath-Ausdrücken

2.7.5 Operatoren

Innerhalb von XPath-Ausdrücken können verschiedene Operatoren verwendet werden. Die nachstehenden Tabellen geben eine Übersicht über die möglichen mathematischen, booleschen und Ausdrucksoperatoren (vgl. [TIDWELL 2002], S. 341-343).

Operator	Erläuterung
+ (plus)	addiert Zahlen miteinander
- (minus)	subtrahiert Zahlen voneinander
* (Multiplikation)	multipliziert Zahlen miteinander
div	dividiert Zahlen miteinander
mod	gibt den ganzzahligen Rest einer Division zurück

Tabelle 2: Mathematische Operatoren in XPath

Operator	Erläuterung
=	prüft, ob der vorhergehende Ausdruck gleich dem nachfolgenden ist
<	prüft, ob der vorhergehende Ausdruck kleiner als der nachfolgende ist (muss als <code>&lt;</code> geschrieben werden)
<=	prüft, ob der vorhergehende Ausdruck kleiner oder gleich dem nachfolgenden ist (muss als <code>&lt;=</code> geschrieben werden)
>	prüft, ob der vorhergehende Ausdruck größer als der nachfolgende ist (muss gegebenenfalls als <code>&gt;</code> geschrieben werden)
>=	prüft, ob der vorhergehende Ausdruck größer oder gleich dem nachfolgenden ist (muss gegebenenfalls als <code>&gt;=</code> geschrieben werden)
!=	prüft, ob der vorhergehende Ausdruck ungleich dem nachfolgenden ist
and	prüft, ob der vorhergehende und der nachfolgende Ausdruck den booleschen Wert <code>true</code> zurückliefert

Operator	Erläuterung
or	prüft, ob der vorhergehende oder der nachfolgende Ausdruck den booleschen Wert <code>true</code> zurückliefert

Tabelle 3: Boolesche Operatoren in XPath

Operator	Erläuterung
/	Schrittoperator – trennt die einzelnen Schritte (Lokalisierungsschritte) im Lokalisierungspfad voneinander bzw. wählt das Wurzelement, wenn am Beginn eines Ausdrucks
//	verkürzte Schreibweise für die <code>descendant-or-self</code> -Achse
.	verkürzte Schreibweise für die <code>self</code> -Achse
..	verkürzte Schreibweise für die <code>parent</code> -Achse
@	verkürzte Schreibweise für die Attribute
	Vereinigungsoperator – vereinigt mehrere Ausdrücke miteinander
*	Platzhalter – liefert alle Knoten des Kontextknotens
[]	Prädikatoperator – enthält ein oder mehrere Prädikate zur Filterung des Ausdrucksergebnisses
\$	Variablenoperator – kennzeichnet eine Variable bzw. ruft diese auf

Tabelle 4: Ausdrucksoperatoren in XPath

2.8 XSL-FO

XSL-FO oder auch Extensible Stylesheet Language – Formatting Objects gehört wie XSLT und XPath ebenfalls zur XSL-Familie und ist ihrerseits wiederum eine XML-Anwendung.

XSL-FO-Dokumente, die so genannten XSL-FO-Stylesheets, beinhalten Anweisungen, mit denen die Daten eines XML-Dokuments für deren Ausgabe formatiert werden. Die Anweisungen ermöglichen unter anderem die exakte Positionierung von Textblöcken, Grafiken und anderen

Layoutelementen sowie die grundlegende Einrichtung der einzelnen Ausgabeseiten. Da XSL-FO zur Erzeugung von hochwertigen Druckerzeugnissen entwickelt wurde, ermöglicht sie neben dem reinen Positionieren der Elemente auch umfangreiche Eingriffe in die Typografie, Einbindung von Farbprofilen, Festlegung von verschiedenen Seitenlayouts und Seitenfolgen, automatische Nummerierung von Seiten, Erzeugung von mehrspaltigen Satzspiegeln sowie die Erzeugung typischer Elemente wie Absätze, Listen, Tabellen, deren Formatierung und Einbindung von Grafiken.

XSL-FO-Dokumente können zwar grundsätzlich händisch erzeugt werden, in der Regel werden sie jedoch, meist als notwendiger Zwischenschritt zum druckfähigen Ausgabedokument, mit einem XSL-Stylesheet und dem XML-Quelldokument über eine XSLT-Transformation erstellt (vgl. [SKULSCHUSS 2005], S. 15). Das XSL-FO-Dokument wird also aus dem XML-Quelldokument als Datenquelle gemeinsam mit einem XSL-Stylesheet über einen XSLT-Prozessor erzeugt. In einem nächsten Verarbeitungsschritt wird das XSL-FO-Dokument dann durch einen FO-Prozessor in das Zielformat, beispielsweise das PDF-Format, transformiert. Die nachstehende Abbildung zeigt das grundlegende Prinzip dieser Transformationskette.

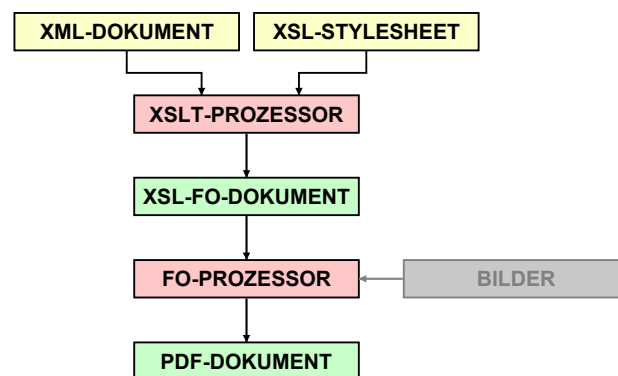


Abbildung 16: Prinzip der Erzeugung eines PDF-Dokuments

2.8.1 Grundgerüst eines XSL-FO-Stylesheets

XSL-FO-Dokumente sind wohlgeformte XML-Dokumente und beginnen als solche mit der XML-Deklaration. Das Wurzelement eines XSL-FO-Stylesheets ist das `<fo:root>`-Element und beinhaltet alle weiteren XSL-FO-Anweisungen. In ihm wird auch der Namensraum der FO-Elemente deklariert. Das nachfolgende Listing zeigt das Grundgerüst eines XSL-FO-Stylesheets.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <fo:root
03      xmlns:fo="http://www.w3.org/1999/XSL/Format">
04      <fo:layout-master-set>
05          <fo:simple-page-master master-name="...">
06              <fo:region-body/>
07          </fo:simple-page-master>
08      </fo:layout-master-set>
09
10      <fo:page-sequence
11          master-reference="...">
12          <fo:flow flow-name="...">
13              ...
14          </fo:flow>
15      </fo:page-sequence>
16  </fo:root>
```

Listing 10: Grundgerüst eines XSL-FO-Stylesheets

2.8.2 Seitenlayout

Innerhalb des `<fo:layout-master-set>`-Elements werden mit den `<fo:simple-page-master>`-Elementen die Seitenvorlagen definiert. Dieses Element beinhaltet das `master-name`-Attribut, mit dem die jeweilige Seitenvorlage über ihren Namen im Stylesheet aufgerufen und benutzt werden kann. Das nachfolgende Listing zeigt die Definition einer Seitenvorlage für eine hochformatige DIN A4 Seite.

```
01  <fo:layout-master-set>
```

```
02     <fo:simple-page-master
03         master-name="Inhaltsseiten"
04         page-width="210mm"
05         page-height="297mm"
06         margin="5mm">
07         <fo:region-body/>
08     </fo:simple-page-master>
09 </fo:layout-master-set>
```

Listing 11: Definition einer Seitenvorlage

Über die `page-width`- und `page-height`-Attribute werden die Ausgabemaße der Seite definiert, das `margin`-Attribut legt die Ränder des Druckbereichs fest. Dieses Attribut kann, wie im obigen Listing in Kurzschreibweise notiert werden, es können aber auch vier separate Attribute (`margin-top`, `margin-right`, `margin-bottom` und `margin-left`) für jeden Rand angegeben werden. Auch in der Kurzschreibweise kann, genau wie bei CSS, für jeden Rand ein einzelner Wert angegeben werden. Dabei repräsentieren die Werte die Ränder im Uhrzeigersinn, beginnend beim oberen Rand. Das `<fo:region-body>`-Element definiert einen der Bereiche des Ausgabedokuments, die im folgenden Abschnitt erläutert werden.

2.8.3 Seitenbereiche

Eine Seite des Ausgabedokuments ist grundsätzlich in fünf Bereiche

- `region-body`
- `region-before`
- `region-after`
- `region-start`
- `region-end`

aufgeteilt. Diese sind in der Grundeinstellung nach der westlichen Schreibrichtung von links nach rechts wie in der nachstehenden Abbildung aufgeteilt.

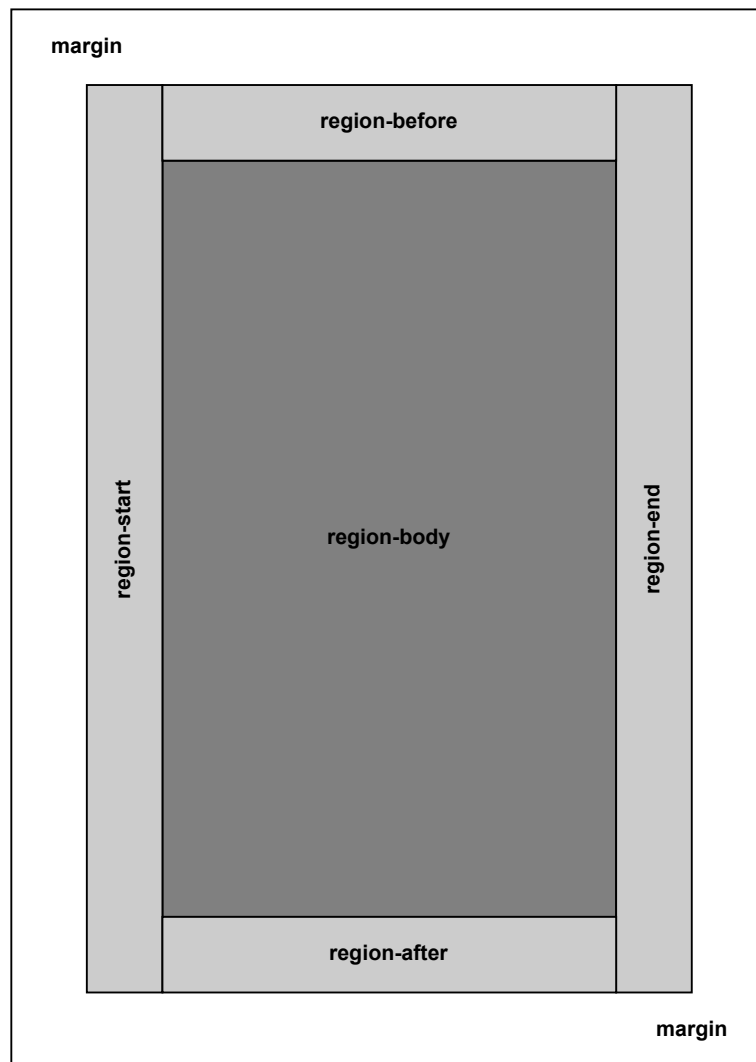


Abbildung 17: Seitenbereiche

In der Regel werden die Randbereiche `region-before`, `-after`, `-start` und `-end` für statische Inhalte wie Logos, Seitenzahlen oder Adressen verwendet, wogegen der eigentliche Seiteninhalt im Bereich `region-body` ausgegeben wird. Der Abstand des druckbaren Bereichs vom Seitenrand kann wiederum, wie im vorherigen Abschnitt aufgeführt, über `margin`-Attribute im `<fo:simple-page-master>`-Element festgelegt werden. Werden neben dem benötigten `<fo:region-body>`-Element die optionalen Elemente für die Randbereiche verwendet, ist bei deren Definition die Reihenfolge wichtig, um ein gültiges XSL-FO-Stylesheet zu erhalten. Diese Reihenfolge bedingt, dass der obere bzw. untere Randbereich nicht die komplette Seitenbreite einnimmt, sondern vom linken und rech-

ten Randbereich in seiner Breite beschränkt wird. Sollen der obere oder untere Randbereich dagegen die komplette Seitenbreite einnehmen, kann dies über das `precedence`-Attribut mit dem Wert `true` innerhalb der Definition dieser Bereiche erzielt werden.

Die einzelnen Bereiche können ihrerseits mit diversen Attributen versehen und weiter in Blöcke segmentiert werden, was ein sehr fein konstruiertes Layout ermöglicht.

Das nachstehende Listing zeigt die Definition der Randbereiche mit einigen Attributen.

```
01  <fo:layout-master-set>
02    <fo:simple-page-master
03      master-name="Inhaltsseiten"
04      page-width="210mm"
05      page-height="297mm"
06      margin="5mm">
07      <fo:region-body
08        margin="40mm 20mm 50mm 30mm"/>
09      <fo:region-before
10        extent="40mm"
11        background-color="#ff0000"/>
12      <fo:region-after
13        extent="50mm"
14        background-color="#ff0000"/>
15      <fo:region-start
16        extent="30mm"
17        background-color="#ff0000"/>
18      <fo:region-end
19        extent="20mm"
20        background-color="#ff0000"/>
21    </fo:simple-page-master>
22  </fo:layout-master-set>
```

Listing 12: Definition der Randbereiche

Die `extent`-Attribute innerhalb der Randelemente definieren deren Ausdehnung, die `background-color`-Attribute deren Hintergrundfarbe.

Die Werte des `margin`-Attributs im `<fo:region-body>`-Element entsprechen den Ausdehnungen der Ränder und sorgen dafür, dass der eigentliche Inhaltsbereich nicht mit den Randbereichen überlappt.

Zusätzlich können die Seitenbereiche über das `region-name`-Attribut mit einem selbstdefinierten Namen versehen werden, um diese später damit anzusprechen. Oftmals ist die Verwendung eines Namens wie `kopfsteg` verständlicher und kürzer als die Ansprache des Elements über die Bezeichnung `xsl-region-before`.

2.8.4 Mehrspaltige Satzspiegel

Wie eingangs erwähnt, können mit XSL-FO auch mehrspaltige Satzspiegel, wie man sie etwa aus Zeitschriften kennt, erzeugt werden. Dafür muss lediglich die Anzahl der Spalten und der Abstand der Spalten zueinander angegeben werden (vgl. [SKULSCHUS 2005], S. 238). Dies wird, wie im folgenden Listing ersichtlich, mit dem `column-count`- und dem `column-gap`-Attribut innerhalb des `<fo:region-body>`-Elements erreicht.

```
01 <fo:region-body ... column-count="3" column-  
    gap="5mm" />
```

Listing 13: Erzeugen eines dreispaltigen Satzspiegels

2.8.5 Schreibrichtung und Textausrichtung

Die erwähnte Leserichtung von links nach rechts kann für Dokumente in Sprachen, deren Schreibrichtung von der westlichen abweicht, ohne weiteres abgeändert werden. Mit dem `writing-mode`-Attribut innerhalb des `<fo:simple-page-master>`-Elements bzw. innerhalb einzelner Textblöcke kann die Schreibrichtung festgelegt werden. Zulässige Werte des `writing-mode`-Attributs sind `lr-tb` (von links nach rechts - von oben nach unten), `rl-tb` (von rechts nach links - von oben nach unten) und `tb-rl` (von oben nach unten - von rechts nach links). Eine weitere Möglichkeit, die Ausrichtung des Textes zu beeinflussen, bietet das `reference-orientation`-Attribut. Mit ihm kann die Ausrichtung eines

Textblocks in 90°-Schritten variiert werden. Durch die Verwendung beider Attribute ergeben sich insgesamt zwölf verschiedene Schreibrichtungen, wie die nachstehende Abbildung verdeutlicht.

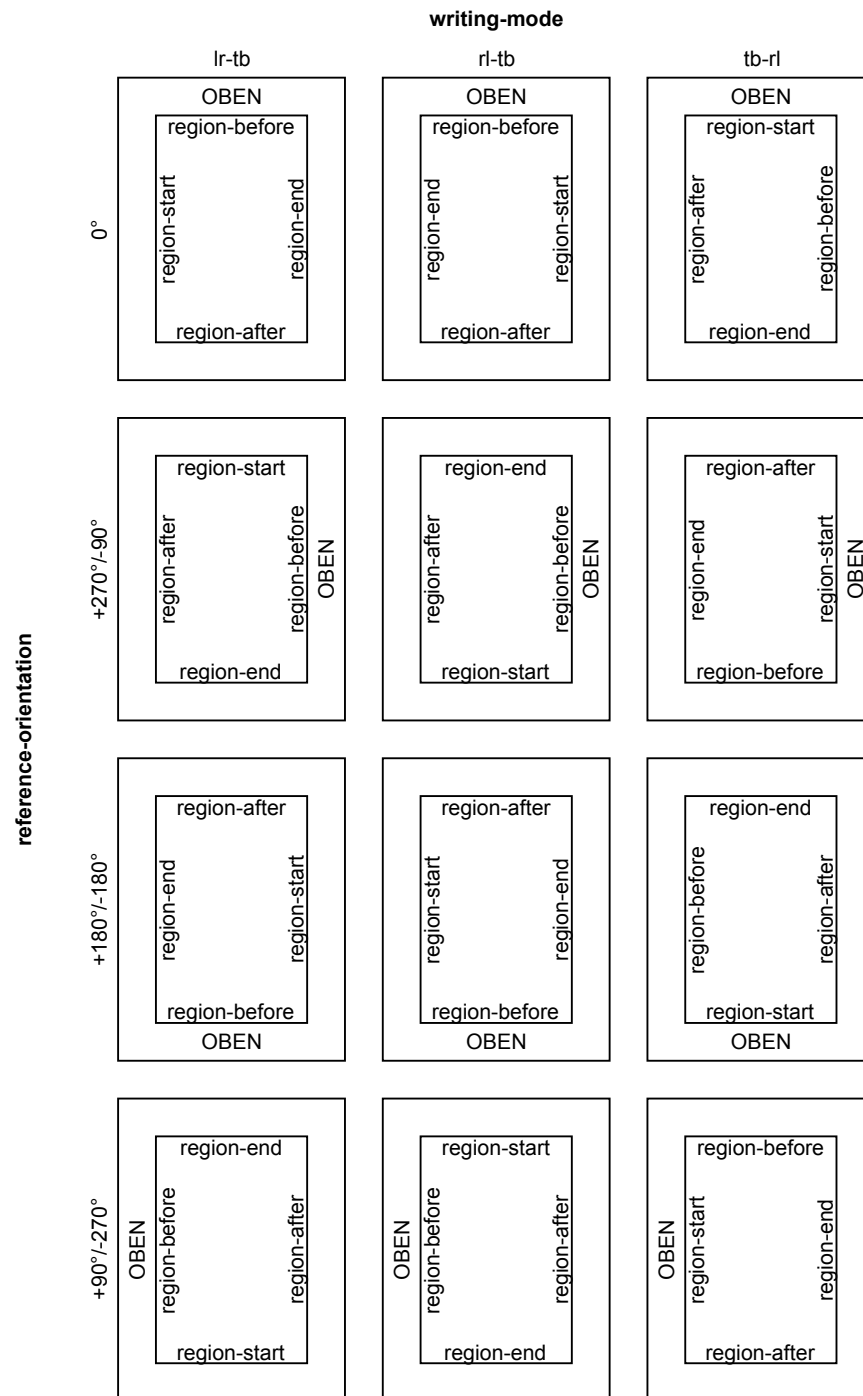


Abbildung 18: mögliche Schreibrichtungen und Ausrichtungen (nach [DATA2TYPE 1])

2.8.6 Elementstruktur

Neben dem sehr einfachen Aufbau des obigen Beispiels kann ein XSL-FO-Stylesheet wesentlich umfangreichere Anweisungen enthalten. Eine Übersicht über die möglichen Elemente zeigt die nachfolgende Abbildung.

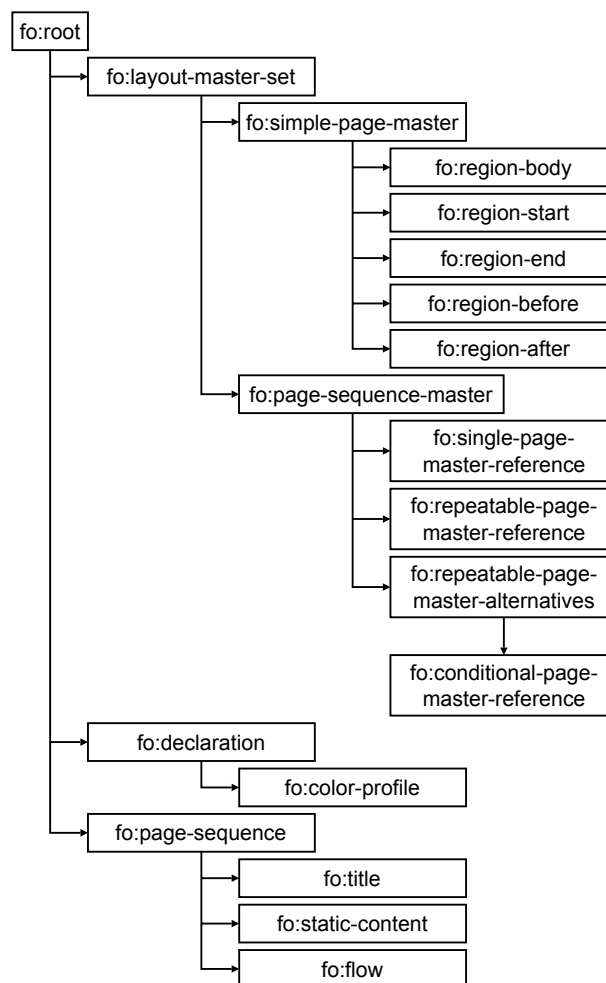


Abbildung 19: Elementstruktur eines XSL-FO-Stylesheets (nach [DATA2TYPE 2])

2.8.7 Inhalt

Nachdem die Seitenvorlagen definiert sind, kann der eigentliche Inhalt für das Ausgabedokument eingefügt werden. Dafür wird das `<fo:page-sequence>`-Element und dessen Kindelemente `<fo:static-content>` und `<fo:flow>` benötigt. Das `<fo:static-content>`-Element

enthält statische Inhalte, wie beispielsweise Seitenzahlen oder Kolummentitel. Das `<fo:flow>`-Element darf nur einmal existieren und enthält den Fließtext des Dokuments, welcher, wenn nötig, auf mehrere Seiten oder Spalten verteilt wird. Das nachfolgende Listing zeigt die Definition einer kompletten Seite in einem XSL-FO-Stylesheet. Dabei werden die einzelnen Bereiche mit selbstdefinierten Namen angesprochen, wie in *Abschnitt 2.8.8* erwähnt wird.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/
    Format">
03      <fo:layout-master-set>
04          <fo:simple-page-master
05              master-name="Inhaltsseiten"
06              page-width="210mm" page-height="297mm"
07              margin="5mm">
08              <fo:region-body
09                  margin="40mm 20mm 50mm 30mm"
10                  region-name="inhalt"/>
11              <fo:region-before
12                  extent="40mm" region-name="kopfsteg"/>
13              <fo:region-after
14                  extent="50mm" region-name="fusssteg"/>
15              <fo:region-start
16                  extent="30mm" region-name="bundsteg"/>
17              <fo:region-end
18                  extent="20mm" region-name="aussensteg"/>
19          </fo:simple-page-master>
20      </fo:layout-master-set>
21      <fo:page-sequence
22          master-reference="Inhaltsseiten">
23          <fo:static-content flow-name="kopfsteg">
24              <fo:block
25                  text-align-last="justify"
26                  margin-top="10mm"
27                  border-bottom="0.25mm solid #000"
28                  font-weight="bold">
29              Das Hallo Welt Buch
```

```
30         <fo:leader/>
31         <fo:page-number/>
32     </fo:block>
33 </fo:static-content>
34 <fo:static-content flow-name="fusssteg">
35     <fo:block
36         text-align-last="right"
37         font-weight="bold" margin-top="20mm"
38         border-top="0.25mm solid #000">
39         Dipl.-Ing. (FH) Patrick Henkel
40     </fo:block>
41 </fo:static-content>
42 <fo:flow flow-name="inhalt">
43     <fo:block font-weight="bold">Erstes Kapi-
44 tel</fo:block>
45     <fo:block>
46         Diese Buch handelt von einer allseits be-
47 liebten Aussage: Hallo Welt!
48     </fo:block>
49 </fo:flow>
50 </fo:page-sequence>
51 </fo:root>
```

Listing 14: Definition einer kompletten Seite in einem XSL-FO-Stylesheet

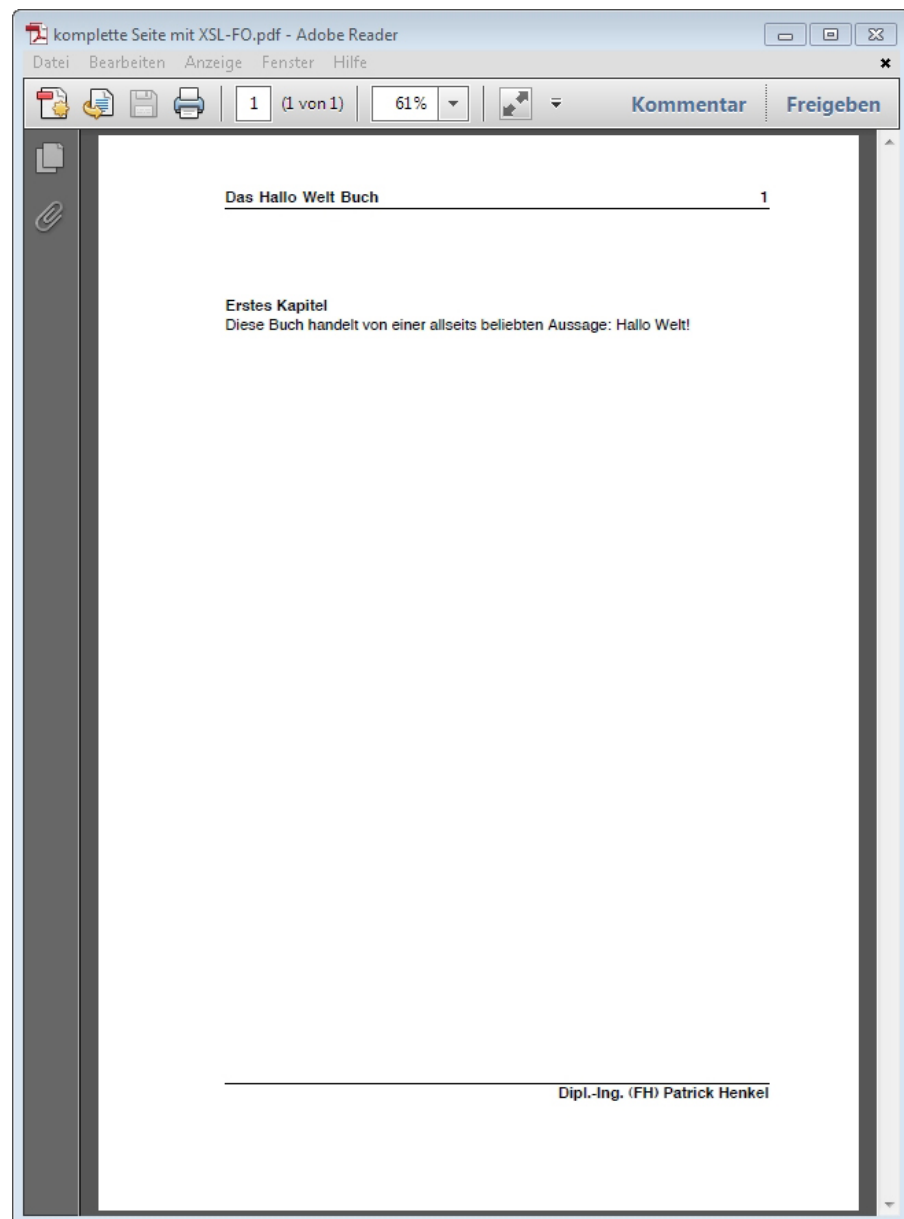


Abbildung 20: Ausgabe der PDF im Adobe Reader

2.8.8 Festlegen von Seitenfolgen

Sollen in Ergebnisdokumenten verschiedene Seitenvorlagen in einer bestimmten Reihenfolge verwendet werden, beispielsweise für Titelseite, linke und rechte Seiten und Schlussblatt eines Kapitels in einem Buch, so werden diese Seitenfolgen im `<fo:page-sequence-master>`-Element festgelegt. Dieses Element wird über das `master-name`-Attribut mit einem

eindeutigen Namen versehen, über den es angesprochen werden kann. Es wird, wie die Seitenvorlagen selbst, im `<fo:layout-master-set>`-Element definiert und hat die möglichen Kindelemente `<fo:single-page-master-reference>` (für einzelne Seiten), `<fo:repeatable-page-master-reference>` (für eine sich wiederholende Seitenvorlage) und `<fo:repeatable-page-master-alternatives>` (für mehrere sich wiederholende Seitenvorlagen). Jedes dieser Elemente ruft eine vorher definierte Seitenvorlage über das `master-reference`-Attribut auf. Zusätzlich verfügen die Elemente `<fo:repeatable-page-master-reference>` und `<fo:repeatable-page-master-alternatives>` über das optionale Attribut `maximum-repeats`, mit dem die Anzahl der Wiederholungen bestimmt werden kann.

Das `<fo:repeatable-page-master-alternatives>`-Element verfügt seinerseits wiederum über das Kindelement `<fo:conditional-page-master-reference>`, dessen Attribute `page-position`, `odd-or-even` und `blank-or-not-blank` festlegen, an welcher Stelle die entsprechende Seitenvorlage verwendet werden soll. So kann mit dem Attribut `page-position="first"` festgelegt werden, dass die Seitenvorlage nur auf die erste Seite der Seitenfolge angewendet wird. Mit dem Attribut `odd-or-even="odd"` wird bestimmt, dass die entsprechende Seitenvorlage nur auf ungerade Seiten der Seitenfolge angewendet wird. Das Attribut `blank-or-not-blank="blank"` legt fest, dass eine leere Seite eingefügt wird, wenn die letzte Seite der Seitenfolge eine ungerade Seite sein soll, die Länge des Textes aber nicht ausreicht, diese letzte Seite zu füllen.

Das folgende Listing zeigt die Definition einer Seitenfolge für ein Kapitel, dessen erste Seite auf einer ungeraden Seite mit der Seitenvorlage `Inhalt-Deckblatt` beginnen soll, die letzte Seite auf einer geraden Seite mit der Seitenvorlage `Inhalt-Schlussblatt` enden soll und auf dessen übrigen Seiten je nach Position jeweils die Seitenvorlagen `Inhalt-Links` und `Inhalt-Rechts` angewendet werden sollen.

```
01 <fo:page-sequence-master  
02   master-name="Kapitel">  
03   <fo:repeatable-page-master-alternatives>
```

```

04     <fo:conditional-page-master-reference
05         master-reference="Inhalt-Deckblatt"
06         page-position="first"
07         odd-or-even="odd" />
08     <fo:conditional-page-master-reference
09         master-reference="Inhalt-Links"
10         page-position="rest"
11         odd-or-even="even" />
12     <fo:conditional-page-master-reference
13         master-reference="Inhalt-Rechts"
14         page-position="rest"
15         odd-or-even="odd" />
16     <fo:conditional-page-master-reference
17         master-reference="Inhalt-Schlussblatt"
18         page-position="last"
19         odd-or-even="even"
20         blank-or-not-blank="blank" />
21     <fo:conditional-page-master-reference
22         master-reference="Inhalt-Links"
23         page-position="last"
24         odd-or-even="even" />
25     <fo:conditional-page-master-reference
26         master-reference="Inhalt-Rechts"
27         page-position="last"
28         odd-or-even="odd" />
29 </fo:repeatable-page-master-alternatives>
30 </fo:page-sequence-master>

```

Listing 15: Seitenfolgen mit verschiedenen Seitenvorlagen

2.8.9 Maßeinheiten

XSL-FO unterstützt eine Reihe von Maßeinheiten, um Positionen, Maße, Schriftgrößen usw. festzulegen. Diese Einheiten sind in der nachfolgenden Tabelle aufgelistet.

Maßeinheit	Erläuterung
cm	Zentimeter

Maßeinheit	Erläuterung
mm	Millimeter
in	Inch (1 in = 2,54 cm)
pt	DTP-Punkte (1 pt = 1/72 in = 0,353 mm); geringfügig abweichend von Didot-Punkten (= 0,376 mm)
pc	Picas (1 pc = 12 pt)
px	Pixel
em	von der Schriftart und -größe abhängige Einheit (Breite eines m)
%	Prozent

Tabelle 5: von XSL-FO unterstützte Maßeinheiten

2.8.10 Farben

Für die Angabe von Farbwerten stehen in XSL-FO-Stylesheets vier verschiedene Möglichkeiten zur Verfügung:

- Benannte Farben (z.B. `color="red"`)
- RGB (z.B. `color="rgb(255, 0, 0)"`)
- HEX (z.B. `color="#ff0000"`) (*in diesem Beispiel wäre auch die verkürzte Schreibweise `color="#f00"` möglich*)
- CMYK (z.B. `color="cmyk(0, 1, 1, 0)"`)

2.8.11 Blockelemente

Das zentrale Gestaltungsmittel in XSL-FO sind `<fo:block>`-Elemente. In gewisser Hinsicht ähnelt dieses Konzept den `<div>`-Elementen in (X)HTML. Auch in XSL-FO können diese Elemente mit Attributen versehen werden und somit formatiert werden. Diese Blockelemente können beliebig ineinander verschachtelt werden und dienen beispielsweise als Rahmen für Absätze, Abbildungen, Listen und Tabellen. Um beispielsweise eine Überschrift und einen darauffolgenden Absatz auszugeben, werden diese beiden Inhaltselemente jeweils in einem `<fo:block>`-Element definiert und mit den entsprechenden Formatierungsattributen versehen. Ein Beispiel dafür zeigt das nachfolgende Listing.

```
01 <fo:block
02     font-family="Times"
03     font-size="18pt"
04     font-weight="bold"
05     space-after="1em"
06     color="#6666ff">
07     Erstes Kapitel
08 </fo:block>
09 <fo:block
10     font-family="Helvetica"
11     font-size="12pt"
12     space-after="1em">
13     Diese Buch handelt von einer allseits beliebten
14     Aussage: Hallo Welt!
15 </fo:block>
```

Listing 16: Blockelemente zur Formatierung der Inhalte

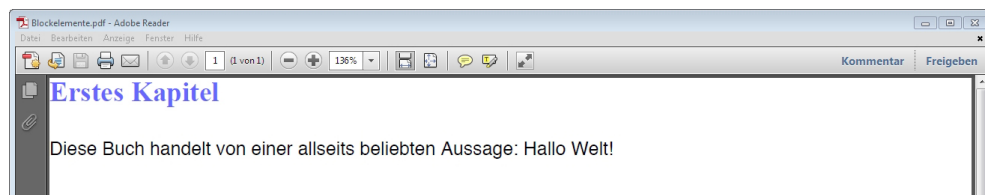


Abbildung 21: Ausgabe formatierter Blockelemente im Adobe Reader

2.8.12 Listen

Für die Ausgabe von Listen mit XSL-FO wird das `<fo:list-block>`-Element (Listenblock) mit dessen Kindelement `<fo:list-item>` (Listenelement) und wiederum dessen Kindelementen `<fo:list-item-label>` (Symbol des Listenelements) und `<fo:list-item-body>` (Inhalt des Listenelements) verwendet. XSL-FO unterscheidet dabei nicht zwischen geordneten und ungeordneten Listen. Sollen geordnete Listen ausgegeben werden, müssen die entsprechenden Nummerierungen entweder händisch oder über XSLT erzeugt werden. Für ungeordnete Listen kann dagegen jedes beliebige Symbol, beispielsweise eine Grafik oder eine Glyphen einer beliebigen Schriftfamilie, verwendet werden.

Zur sinnvoll formatierten Ausgabe einer Liste ist die Verwendung von verschiedenen Attributen zur Definition der Abstände unumgänglich, da sich sonst der Inhalt des Listenelements und dessen Symbol überlagern, wie die nachstehende Abbildung zeigt.

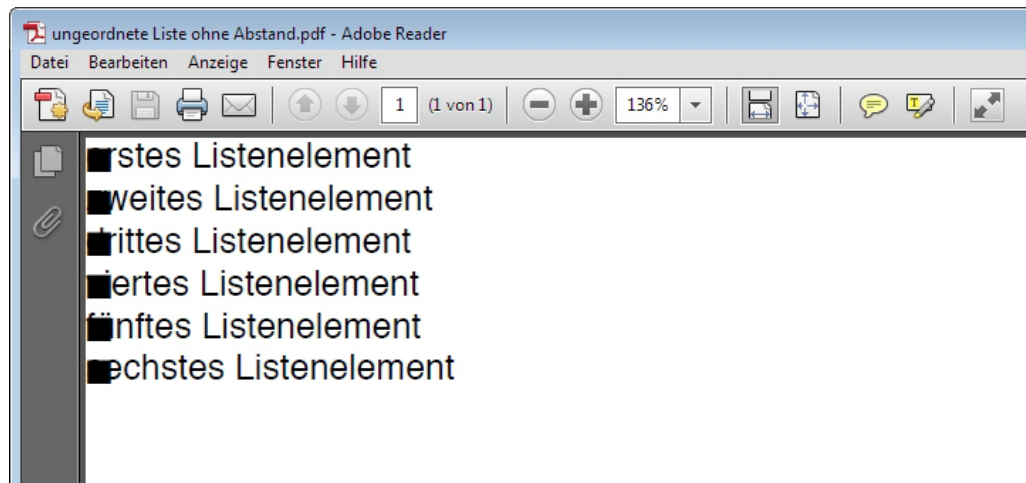


Abbildung 22: Ausgabe einer ungeordneten Liste ohne Abstandsdefinition

Im einfachsten Fall kann dem `<fo:list-item-body>`-Element mit dem Attribut `start-indent` ein Abstand vom linken Rand der Liste gegeben werden.

Das nachfolgende Listing zeigt die Erzeugung einer ungeordneten Liste mit einem gefüllten Quadrat aus der Arial-Schriftfamilie und dem `start-indent`-Attribut zur Festlegung eines Abstands von 15 mm des Listenelementinhalts vom linken Listenrand.

```

01  <fo:list-block>
02    <fo:list-item>
03      <fo:list-item-label>
04        <fo:block>&#9632;</fo:block>
05      </fo:list-item-label>
06      <fo:list-item-body
07        start-indent="15mm">
08        <fo:block>erstes Listenelement</fo:block>
09      </fo:list-item-body>
10    </fo:list-item>
11

```

```

12  <fo:list-item>
13      <fo:list-item-label>
14          <fo:block>&#9632;</fo:block>
15      </fo:list-item-label>
16      <fo:list-item-body
17          start-indent="15mm">
18          <fo:block>zweites Listenelement</fo:block>
19      </fo:list-item-body>
20  </fo:list-item>
21  ...
22  <fo:list-item>
23      <fo:list-item-label>
24          <fo:block>&#9632;</fo:block>
25      </fo:list-item-label>
26      <fo:list-item-body
27          start-indent="15mm">
28          <fo:block>sechstes Listenelement</fo:block>
29      </fo:list-item-body>
30  </fo:list-item>
31  </fo:list-block>

```

Listing 17: ungeordnete Liste mit Attribut zur Definition des Abstands

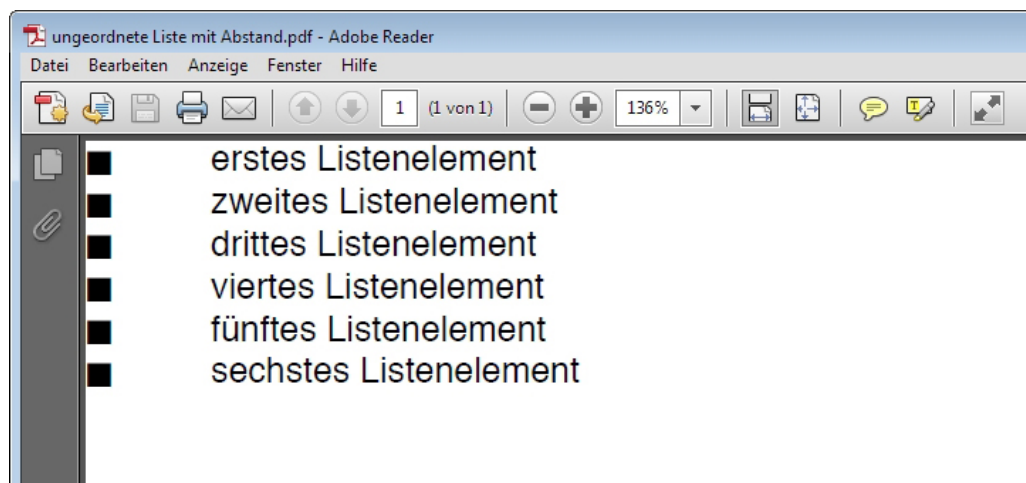


Abbildung 23: Ausgabe einer ungeordneten Liste mit Abstandsdefinition

2.8.13 Grafiken

Grafiken können, ähnlich wie in (X)HTML, entweder als Block- bzw. Inlinenelement eingebunden werden oder als Hintergrundgrafik eines Blockelements dienen.

Im ersten Fall werden sie über das `<fo:external-graphics>`-Element im XSL-FO-Stylesheet definiert. Dieses Element verfügt wiederum über diverse Attribute, mit denen das Erscheinungsbild der Grafik festgelegt wird. Im `src`-Attribut wird der Pfad zur anzuzeigenden Grafik bestimmt. Mit den Attributen `content-height` und `content-width` werden die Maße der Grafik, die diese im Ausgabedokument haben soll, definiert. Werden diese Werte nicht gesetzt, wird die Grafik in ihren Originalmaßen ausgegeben. Wird nur einer der Werte gesetzt, passt sich der jeweils andere Wert proportional an. Neben numerischen Angaben der Maße mit den im *Abschnitt 2.8.9* aufgeführten Einheiten kann der Wert dieser Attribute auch `scale-to-fit` sein. In diesem Fall passt sich die Grafik an das übergeordnete Blockelement an. Das nachfolgende Listing zeigt die Einbindung einer externen Grafik.

```
01 <fo:external-graphics
02   src="abbildung01.jpg"
03   content-width="50mm" />
```

Listing 18: Einbindung einer Grafik mit expliziter Angabe der Breite

Soll eine Grafik als Hintergrund eines Blockelements verwendet werden, so wird dies über das Attribut `background-image` des entsprechenden `<fo:block>`-Element erreicht. Neben der Angabe des Pfads zur Grafik kann das Erscheinungsbild über weitere Attribute, wie `background-position` (Positionierung der Hintergrundgrafik im Blockelement) oder `background-repeat` (Kachelung der Hintergrundgrafik im Blockelement) beeinflusst werden. Das nachfolgende Listing zeigt die Ausgabe einer Grafik als Hintergrundgrafik eines Blockelements ohne Kachelung mit jeweils 10 mm Abstand vom oberen und linken Rand.

```
01 <fo:block
02   background-image="abbildung01.jpg"
```

```
03     background-position="10mm"  
04     background-repeat="no-repeat">  
05     ...  
06 </fo:block>
```

Listing 19: Einbindung einer Grafik als Hintergrundgrafik eines Blockelements

2.8.14 Hyperlinks

Hyperlinks in einem XSL-FO-Stylesheet können entweder auf Ziele innerhalb des Dokuments, beispielsweise als Querverweise zu anderen Kapiteln oder auf externe Ziele, beispielsweise auf URLs von Websites, verweisen. Dafür wird das `<fo:basic-link>`-Element mit entsprechenden Attributen verwendet. Mit dem Attribut `internal-destination` wird auf ein Ziel innerhalb des Dokuments in Form einer ID verwiesen. Diese ID muss dem Element entweder händisch zugeordnet werden oder beispielsweise über die XSLT-Funktion `generate-id()` automatisch erzeugt werden. Mit dem Attribut `external-destination` wird dagegen auf ein externes Ziel in Form einer URL oder einer Pfadangabe verwiesen. Mit dem Attribut `show-destination` kann bestimmt werden, ob sich das Ziel im gleichen Fenster (`show-destination="replace"`) oder in einem neuen Fenster öffnen soll (`show-destination="new"`). Das nachfolgende Listing zeigt die Verwendung von Hyperlinks in einem XSL-FO-Stylesheet.

```
01 <fo:block  
02     id="kap1">  
03     Erstes Kapitel  
04 </fo:block>  
05  
06 <fo:block>  
07     <fo:basic-link  
08         internal-destination="kap1">  
09         siehe Kapitel 1  
10     </fo:basic-link>  
11  
12     <fo:basic-link
```

```
13         external-destination="http://www.patrick-hen-
kel.net">
14         www.patrick-henkel.net
15     </fo:basic-link>
16 </fo:block>
```

Listing 20: Hyperlinks zu internen und externen Zielen

2.8.15 Attribut-Sets

Wie in *Abschnitt 2.8.11* beschrieben wurde, können Blockelemente mit Attributen für die Formatierung derer Inhalte (z.B. Schriftfamilie und -größe, Farbe usw.) versehen werden. Wenn bestimmte Blockelemente mit immer wiederkehrenden Formatierungen versehen werden sollen, bietet sich die Verwendung von Attribut-Sets an (vgl. [SKULSCHUS 2005], S. 227). Diese Attribut-Sets sind Sammlungen von Attributen, die gemeinsam auf ein Blockelement mit `xsl:use-attribute-sets="..."` angewendet werden können. Attribut-Sets können allerdings nicht direkt innerhalb von XSL-FO-Stylesheets definiert werden. Sie können nur innerhalb von XSL-Stylesheets, die ein XSL-FO-Dokument erzeugen, genutzt werden.

In gewisser Weise ähnelt die Verwendung dieser Sets der von Formatvorlagen in Textverarbeitungsprogrammen. Der große Vorteil dieser Vorgehensweise erschließt sich spätestens, wenn ein immer wiederkehrendes Attribut in vielen Elementen geändert werden muss. Statt beispielsweise in allen Blockelementen die Schriftgröße von 10 pt auf 11 pt ändern zu müssen, muss diese Änderung nur einmal im Attribut-Set vorgenommen werden. Das folgende Listing demonstriert die Verwendung von Attribut-Sets.

```
01 <xsl:attribute-set name="ueberschrift1">
02     <xsl:attribute name="font-family">
03         Arial
04     </xsl:attribute>
05     <xsl:attribute name="font-size">
06         20pt
```

```
07     </xsl:attribute>
08     <xsl:attribute name="font-weight">
09         bold
10     </xsl:attribute>
11 </xsl:attribute-set>
12 ...
13 <fo:block xsl:use-attribute-sets="ueberschrift1">
14     ...
15 </fo:block>
```

Listing 21: Verwendung von Attribut-Sets in XSL-Stylesheets zur FO-Erzeugung

2.9 Neuerungen in XSLT 2.0 und XPath 2.0

Wie in *Abschnitt 2.6* und *Abschnitt 2.7* angemerkt, liegen sowohl XSLT als auch XPath aktuell in der Version 2.0 vor. Insbesondere auf die in XSLT 2.0 hinzugekommenen Erweiterungen `xsl:result-document` und `xsl:function` soll im Folgenden eingegangen werden. Beide Erweiterungen kamen bei der Entwicklung des Benutzerhandbuchs zum Einsatz und verringerten den Entwicklungsaufwand enorm.

Weitere Neuerungen sind beispielsweise die nun mögliche Ausgabemethode im XHTML-Format, wogegen bisher lediglich die Ausgabemethoden HTML, XML und Text unterstützt wurden und die Rückgabe von Nodesequenzen statt der bisherigen Nodesets.

2.9.1 `xsl:result-document`

Diese Erweiterung ermöglicht es, mit einer einzigen Transformation beliebig viele Ergebnisdokumente zu erzeugen. Dabei können die Ergebnisdokumente vom Format *text*, *xml*, *html* und *xhtml* sein. Speziell bei der Ausgabe von Zieldokumenten im XHTML-Format ist diese Erweiterung ein nicht zu unterschätzender Vorteil. So kann im XSL-Stylesheet beispielsweise ein Template erstellt werden, mit dem für jedes dem Template zugeordneten Element ein neues XHTML-Dokument erzeugt wird. Folgendes Beispiel soll das Vorgehen dabei demonstrieren.

```
01 <xsl:template match="kapitel">
02   <xsl:result-document href="{titel}.html"
    method="xhtml"/>
03   <html>
04     <head>
05       <title>
06         <xsl:value-of select="title"/>
07       </title>
08     </head>
09     <body>
10       <xsl:apply-templates/>
11     </body>
12   </html>
13 </xsl:result-document>
14 </xsl:template>
```

Listing 22: mehrere Ergebnisdokumente mit xsl:result-document

Der XSLT-Prozessor wendet bei der Transformation auf jedes `<kapitel>`-Element das Template an und erzeugt für jedes Kapitel eine einzelne XHTML-Datei. Der jeweilige Dateiname entspricht dabei dem Inhalt des `<titel>`-Elements innerhalb des Kapitels. Der Inhalt des `<body>`-Elements in den XHTML-Dateien wird seinerseits wiederum durch die Anwendung weiterer Templates erzeugt.

Da in den meisten Fällen der Titel eines Kapitels nicht den Anforderungen für Dateinamen genügt, er beispielsweise Leerzeichen, Umlaute oder andere Sonderzeichen enthalten könnte, müsste der Dateiname entweder händisch angepasst werden oder die Kapiteltitel von vornherein entsprechend angepasst werden. Da dies wiederum dem Gedanken der Medienneutralität widersprechen würde, liegt der Gedanke nahe, dies in einem automatisierten Prozess durchzuführen. Genau diese Möglichkeit bieten die in XSLT 2.0 benutzerdefinierten Funktionen, die im nächsten Abschnitt erläutert werden.

2.9.2 xsl:function

Neben der Implementierung einer Vielzahl neuer Funktionen (vgl. [W3C 10]) bietet XSLT 2.0 die Möglichkeit, eigene Funktionen zu entwickeln und anzuwenden. Eine solche benutzerdefinierte Funktion wurde im Rahmen der Entwicklung des Benutzerhandbuchs zur Umwandlung der Kapitel- und Abschnittsnamen in gültige Dateinamen erstellt. Das nachfolgende Listing zeigt diese Funktion.

```
01  <xsl:stylesheet
01  ...
01  xmlns:ph="http://www.patrick-henkel.net/XSL/
    ns">
02  ...
01  <xsl:function name="ph:niceTitleInURL">
02  <xsl:param name="string"/>
03  <xsl:value-of select="encode-for-uri(replace
    (replace(replace(replace(replace(lower-
    case(normalize-space($string)), ' ', '_'), '/',
    '-'), 'ä', 'ae'), 'ö', 'oe'), 'ü', 'ue'), 'ß',
    'ss'))"/>
04  </xsl:function>
05  ...
06  </xsl:stylesheet>
```

Listing 23: benutzerdefinierte Funktionen mit `xsl:function`

Diese Funktion wird im XSL-Stylesheet als Kindelement des `<xsl:stylesheet>`-Elements notiert. Das `name`-Attribut im Start-Tag des `<xsl:funtion>`-Elements enthält den Namen der Funktion, hier `niceTitleInURL`. Dafür ist es zusätzlich nötig, einen eigenen Namensraumpräfix im Wurzelement des Stylesheets, dem `<xsl:stylesheet>`-Element zu definieren, hier `xmlns:ph="http://www.patrick-henkel.net/XSL/ns"`. Mit dem Namen der Funktion kann diese in Templates aufgerufen werden. Zusätzlich ist es möglich, den von der Funktion zurückgegebenen Wert über ein optionales `as`-Attribut beispielsweise als `xs:string` oder `xs:integer` zu typisieren. Das `<xsl:param>`-Element instanziiert eine Variable, respektive einen

Parameter, der innerhalb der Funktion verwendet wird. Im konkreten Beispiel dient der Parameter der Übergabe des Kapitel- bzw. Abschnittsnamens an die Funktion. Das `<xsl:value-of>`-Element entspricht in etwa dem `return`-Parameter einer Funktion in anderen Programmiersprachen. Es dient also der Rückgabe des durch die Funktion erzeugten Werts. Im Beispiel werden innerhalb dieses Elements im `select`-Attribut die eigentlichen Umwandlungen des Titels vorgenommen. Die einzelnen Umwandlungsschritte werden von innen nach außen ausgeführt und durchlaufen die folgenden Schritte:

1. *Entfernung eventueller Zeilenumbrüche*
2. *Umwandlung in Kleinschreibung*
3. *Umwandlung von Schrägstrich in Bindestriche*
4. *Umwandlung von Leerzeichen in Unterstriche*
5. *Umwandlung von ä in ae*
6. *Umwandlung von ö in oe*
7. *Umwandlung von ü in ue*
8. *Umwandlung von ß in ss*
9. *„URL-Encoding“ sonstiger Sonderzeichen*

Ausgehend von dem `<title>`-Element

```
01 <title>Das ist ein Text mit Sonderzeichen wie Ä ö  
   Ü und ß</title>
```

wird die Funktion beispielsweise im `select`-Attribut eines `<xsl:value-of>`-Elements aufgerufen

```
01 <xsl:value-of select="ph:niceTitleInURL(title)"/>
```

und erzeugt im Ergebnisdokument die Ausgabe

```
01  <title>das_ist_ein_text_mit_sonderzeichen_wie_ae_
    oe_ue_und_ss</title>
```

2.10 Kommandozeilenbasiertes Transformieren

Falls die genutzte Entwicklungsumgebung keine Möglichkeit zur automatisierten Transformation in die jeweiligen Zielformate bietet, kann diese selbstverständlich auch händisch über die Kommandozeile oder über selbst erstellte Batch-Dateien erfolgen. Dafür ist neben der Quell-XML-Datei und den entsprechenden XSL-Stylesheets lediglich die Installation der Java-Runtime und der XSLT- und FO-Prozessoren erforderlich (siehe [JAVA-RUNTIME], [SAXON HE] und [APACHE FOP]). Im Folgenden soll nun die Vorgehensweise dafür beschrieben werden.

2.10.1 Vorbereitung

Zunächst sollte ein Arbeitsverzeichnis `C:\transform` mit den Unterverzeichnissen `C:\transform\fop` und `C:\transform\saxon` erstellt werden. Weiterhin müssen die benötigten XSLT- bzw. FO-Prozessoren von den Websites der jeweiligen Anbieter heruntergeladen werden.

Das ZIP-Archiv des Saxon HE XSLT-Prozessors enthält lediglich die Datei `saxon9he.jar`. Diese wird in das Verzeichnis `C:\transform\saxon` kopiert.

Das ZIP-Archiv des Apache FOP FO-Prozessors enthält diverse Dateien und Unterordner. Hiervon werden nur die Datei `fop.jar` im Unterordner `build` und sämtliche `.jar`-Dateien im Unterordner `lib` benötigt. Diese werden in das Verzeichnis `C:\transform\fop` kopiert.

Auch das XML-Quelldokument (hier `hallo_welt.xml`) sowie die XSL-Stylesheets zur Transformation in ein XHTML- bzw. PDF-Dokument (hier `hallo_welt_xhtml.xsl` und `hallo_welt_pdf.xsl`) werden in das Verzeichnis `C:\transform` kopiert, so dass sich folgende Verzeichnisstruktur ergibt:

```

└─ transform
   └─ fop
      ├── fop.jar
      └─ ...
   └─ saxon
      ├── saxon9he.jar
      ├── hallo_welt.xml
      ├── hallo_welt_pdf.xsl
      └── hallo_welt_xhtml.xsl
```

Die einzelnen Dateien können der dieser Arbeit beiliegenden CD-ROM entnommen werden.

2.10.2 Händische XSLT-Transformation mit Saxon HE

Um das XML-Quelldokument in ein XHTML-Dokument zu transformieren, muss in der Kommandozeile in das Verzeichnis `C:\transform\saxon` gewechselt und folgender Befehl ausgeführt werden:

```
01  java -jar saxon9he.jar -s:..\hallo_welt.xml
    -xsl:..\hallo_welt_xhtml.xsl -o:..\hallo_welt.
    html
```

Listing 24: Befehl zur Transformation in ein XHTML-Dokument

Der XSLT-Prozessor erzeugt nun das gemäß den Anweisungen im XSL-Stylesheet formatierte XHTML-Dokument.

2.10.3 Händische FO-Transformation mit Apache FOP

Um das XML-Quelldokument in ein PDF-Dokument zu transformieren, muss in der Kommandozeile in das Verzeichnis `C:\transform\fop` gewechselt und folgender Befehl ausgeführt werden:

```
01  java -jar fop.jar -xml ..\hallo_welt.xml -xsl ..\
    hallo_welt_pdf.xsl -pdf ..\hallo_welt.pdf
```

Listing 25: Befehl zur Transformation in ein PDF-Dokument

Der FO-Prozessor erzeugt nun das gemäß den Anweisungen im XSL-Stylesheet formatierte PDF-Dokument.

2.11 Sprachliche Gestaltung von Benutzerhandbüchern

Anleitungen im Allgemeinen und Benutzerhandbücher im Speziellen haben primär eine instruktive und erst in zweiter Linie eine informative Funktion. Das heißt, dass sie den Rezipienten, also den Benutzer, nicht über die Vorzüge des Produkts informieren – das sollte im Idealfall schon durch vorhergehende Marketingmaßnahmen geschehen sein – sondern ihn in der Verwendung des Produkts anleiten und unterstützen sollen. Dazu werden Handlungsanweisungen in kurzen, prägnanten Sätzen formuliert. Wenn nötig, werden diese Handlungsanweisungen durch explizite Anweisungslisten (Schritt-für-Schritt-Anweisungen) untermauert und ergänzt.

Auf Fachbegriffe, die der jeweiligen Zielgruppe nicht unbedingt geläufig sind, sollte im Idealfall verzichtet werden. Wenn deren Verwendung unbedingt nötig ist, sollten diese an geeigneter Stelle erläutert werden. Dies geschieht am besten direkt im Text. Die Verwendung von Fußnoten, Marginalspalten oder Glossaren ist dafür zwar durchaus geeignet, unterbricht jedoch bei zu häufigem Gebrauch den Lesefluss. Weiterhin wirken sich verschachtelte und sehr lange Sätze negativ auf den Lesefluss und das Verständnis des Textes von Benutzerhandbüchern aus.

Zwar sollte auf inhaltliche Redundanzen verzichtet werden, durchaus redundant sollten jedoch die Schemata der Anweisungen sein. Ganz im Gegensatz zu wissenschaftlichen Texten oder Prosa, in denen man für sinngemäß gleiche Sachverhalte verschiedene synonyme Redewendungen nutzen sollte, um den Leser nicht zu langweilen, sollte dies in Benutzerhandbüchern tunlichst unterlassen werden. Hier ist es wesentlich vorteilhafter für gleiche Sachverhalte auch immer die gleichen Redewendungen zu verwenden. Statt

„Klicken Sie auf Schaltfläche A...“

„Betätigen Sie Button B...“

„Aktivieren Sie mit Schaltfläche C...“

sollte sich der verantwortliche Redakteur für eine Schreibweise entscheiden und diese konsequent verwenden. Arbeiten mehrere Redakteure an einem Benutzerhandbuch, empfiehlt sich unter Umständen die Einrichtung einer Terminologiedatenbank, in welcher bestimmte Redewendungen und Begriffe fest definiert werden.

Bei Anweisungen sollen grundsätzlich Aktiv-Positiv-Strukturen verwendet werden, bei denen der Rezipient direkt angesprochen wird. Aussagen wie

„Klicken Sie auf die Schaltfläche A. Das Dateiauswahlfenster öffnet sich.“

haben für den Rezipienten eine wesentlich instruktivere Wirkung als

„Wenn auf die Schaltfläche A geklickt wird, öffnet sich das Dateiauswahlfenster“

Am Positivbeispiel wird auch das vornehmlich zu verwendende Schema *Handlungsaufforderung* → *Handlungsergebnis* deutlich. Der Rezipient wird aufgefordert, eine bestimmte Handlung vorzunehmen und dabei informiert, was seine Handlung auslösen wird.

Neben der rein sprachlichen Gestaltung von Benutzerhandbüchern spielt auch die Nutzung einer nonverbalen Kommunikation eine große Rolle. Insbesondere bei Anweisungen, die die Programmoberfläche einer Software oder ähnlichem betreffen, erleichtert die Verwendung von Symbolen und Grafiken das Verständnis beim Rezipienten. Dabei sollten diese Symbole den Symbolen in der Benutzeroberfläche entsprechen.

Im Allgemeinen sollen Benutzerhandbücher sprachlich einfach, sinnvoll gegliedert und prägnant gestaltet werden.

2.12 Ausgabemedien und Endgeräte

Durch die dem Cross-Media-Publishing eigene Medienneutralität sind der Wahl des Ausgabemediums und der entsprechenden Endgeräte prinzipbedingt kaum Grenzen gesetzt. Allerdings gilt es bei der Entwicklung der jeweiligen Formatierungs- und Transformationsanweisungen zur Erzeugung der Zielformate für die jeweiligen Endgeräte deren spezifische Einschränkungen zu beachten. Denn lediglich die Speicherung der eigentlichen Informationen erfolgt medienneutral, deren Ausgabe ist dagegen in der Regel immer an bestimmte Konventionen hinsichtlich des Formats und des Endgeräts gebunden.

Will man etwa, wie im vorliegenden Fall, als Ausgabemedien den Bildschirm und den Drucker bedienen, bieten sich als Zielformate das (X)HTML-Format und das PDF-Format an. Während (X)HTML-Dokumente im Browser angezeigt werden, kann ein PDF-Dokument sowohl am Bildschirm betrachtet als auch bei Bedarf ausgedruckt werden.

Durch die Verwendung des PDF-Formats gestaltet sich die Entwicklung der Formatierungsanweisungen für Drucker als Endgeräte grundsätzlich eher unkompliziert. Für ein PDF-Dokument werden von vornherein bestimmte Eigenschaften, wie dessen Maße, verwendete Schriftfamilien, -schnitte und -größen definiert und in diesem eingebettet. Diese werden in der Regel, abgesehen von eventuellen Farbabweichungen, von gebräuchlichen Druckern gleich interpretiert und in einer vom Entwickler vorgesehenen Form ausgegeben. Bei der Entwicklung der Formatierungsanweisungen gilt es lediglich eine ansprechende und dem Inhalt gerecht werdende Gestaltung hinsichtlich des Layouts und der Typografie zu entwickeln. Dazu gehört insbesondere eine gut lesbare Schriftgröße und entsprechende Abstände zwischen Absätzen aber auch einzelnen Zeilen und Buchstaben, um eine optimale Lesbarkeit des Textes zu gewährleisten. Der Vorteil des PDF-Formats liegt dabei auf der Hand: der Gestalter des Dokuments kann, entsprechendes Fachwissen vorausgesetzt, eine Vorlage derart gestalten, dass eben diese Kriterien erfüllt werden. Der Benutzer hat dagegen beim Lesen oder Drucken des Dokuments keinen Einfluss auf dessen Darstellung, abgesehen von speziellen Druckeinstellungen wie etwa dem Druck von mehreren Seiten auf einem Blatt o.ä.

und kann keine der Lesbarkeit des Dokuments entgegenwirkenden Einstellungen treffen.

Anders als beim PDF-Format sollten bei der Ausgabe im (X)HTML-Format sowohl die Einstellmöglichkeiten des Browsers als auch die verschiedenen Endgeräte selbst größere Beachtung finden. Bei der Ansicht dieses Formats in einem Browser hat der Benutzer sehr wohl Einfluss auf dessen Darstellung. Er kann beispielsweise die Schriftgröße abändern, unter Umständen sogar eine andere Schriftart wählen oder die CSS-Stylesheets, mit denen die Formatierung des (X)HTML-Dokuments vorgenommen werden, komplett deaktivieren. Auch bei der Verwendung von JavaScript sollte beachtet werden, dass der Benutzer durchaus die Möglichkeit hat (in manchen Unternehmen gilt aus alter Gewohnheit immernoch die Bestimmung, dass JavaScript standardmäßig deaktiviert ist), die Ausführung von JavaScript-Funktionen in seinem Browser zu verbieten und ihm somit bestimmte Funktionen nicht zur Verfügung stehen.

Auch wenn eine Vielzahl der möglichen Eingriffe seitens des Benutzers eher worst-case-Szenarien entsprechen, sollten sie doch bei der Entwicklung der Formatierungsanweisungen in gewissen Grenzen berücksichtigt werden.

Im Allgemeinen sollten bei der Entwicklung der Formatierungsanweisungen die Grundlagen der allgemein anerkannten Usability-Richtlinien berücksichtigt werden. Diese sind unter anderem in der Normenreihe DIN EN ISO 9241, Teil 10 bzw. 110 und 11 ausgiebig definiert.

Betrachtet man die Bandbreite der möglichen Endgeräte, also der Bildschirme der Benutzer, hat sich diese in den vergangenen Jahren enorm vergrößert. Konnte man vor einigen Jahren im Webdesign noch von einer handvoll Standardwerten gebräuchlicher Bildschirmauflösungen ausgehen, hat sich dies spätestens durch die immer größer werdende Anzahl von internetfähigen Mobiltelefonen erübrigt. Große Desktopmonitore, Netbooks, Tablet-Computer und Smartphones können potentiell als Endgeräte in Frage kommen und verbieten durch ihre extrem unterschiedlichen Bildschirmauflösungen nahezu das Festhalten an starren

Größenvorgaben. Seit einiger Zeit verspricht die Anwendung verschiedener Techniken, die sich hinter dem blumigen Buzzword *Responsive Webdesign* verbergen, Abhilfe für dieses Problem. Dieses Thema wird in seinen Grundzügen im folgenden Abschnitt erläutert.

Nicht zuletzt sollten bei der Entwicklung auch die Gewohnheiten und Eigenheiten der Zielgruppe berücksichtigt werden. Ist der Entwickler nicht strikt an ein Corporate Design gebunden oder kann dieses in einem gewissen Rahmen anpassen, lohnt es in jedem Fall, die Gestaltung zielgruppenadäquat vorzunehmen und diese Zielgruppe bereits in der Planungsphase und der Auswahl der Zielformate zu berücksichtigen.

2.13 Responsive Webdesign

Responsive Webdesign, zu deutsch etwa reagierendes Webdesign, ist zunächst nichts anderes, als eine CSS-Technik in Verbindung mit so genannten CSS3 Media Queries ([W3C 11]). Sie ermöglicht, dass das Layout einer Website flexibel ist und sich dynamisch an verschiedene Bildschirmgrößen anpasst, so dass unabhängig vom Endgerät alle (angezeigten) Inhalte erkenn- und lesbar bleiben (vgl. [FLORENCE]).

Während man in der Vergangenheit bereits mit CSS 2.1 unterschiedliche CSS-Stylesheets für unterschiedliche Endgeräte über das `media`-Attribut im `<link>`-Element ansprechen konnte, etwa mit den Werten `screen` (für normale Bildschirme) oder `handheld` (für mobile Endgeräte), funktioniert dies bei modernen Smartphones nicht mehr. Sowohl diese als auch Tablet-Computer ignorieren die Angaben für mobile Endgeräte und weisen sich selbst als normale Bildschirme aus. Dieses Verhalten ist durchaus legitim, denn betrachtet man beispielsweise die Bildschirmgröße eines typischen Tablet-Computers mit 1024 x 768 Pixeln, entspricht diese XGA-Auflösung der bis vor wenigen Jahren noch durchaus üblichen Auflösung herkömmlicher Desktopmonitore. Selbst die Bildschirme von Smartphones erreichen, wie etwa Apples iPhone4 mit 960 x 640 Pixeln, Auflösungen, die denen älterer Desktopmonitore ähneln. Unterschiede finden sich jedoch bekanntermaßen in den physischen Bildschirmgrößen

und damit in der Punktdichte der Bildschirme. Folgendes Rechenbeispiel soll dies verdeutlichen (vgl. [WIKIMEDIA]).

Zunächst wird die Punktdichte eines etwas älteren Monitors mit einer Bilddiagonale von 17 Zoll und einer Auflösung von 1024 x 768 Pixeln bestimmt.

$$ppi = \frac{\sqrt{v_p^2 + h_p^2}}{d_i}$$

ppi : Punktdichte des Bildschirms

h_p : Anzahl der horizontalen Bildpunkte in Pixel

v_p : Anzahl der vertikalen Bildpunkte in Pixel

d_i : Länge der Diagonalen des sichtbaren Bereichs in Inch
(gleichbedeutend mit Zoll)

$$ppi_{\text{monitor}} = \frac{\sqrt{768^2 \text{ Pixel} + 1024^2 \text{ Pixel}}}{17 \text{ Zoll}} \approx \underline{\underline{75ppi}}$$

Berechnet man nun die Punktdichte eines iPad von Apple mit der gleichen Bildschirmauflösung aber einer Bilddiagonalen von lediglich 9,7 Zoll, ergibt sich das folgende Ergebnis.

$$ppi_{\text{ipad}} = \frac{\sqrt{768^2 \text{ Pixel} + 1024^2 \text{ Pixel}}}{9,7 \text{ Zoll}} \approx \underline{\underline{132ppi}}$$

Die Punktdichte des modernen Tablet-Computers ist deutlich höher als die des Desktopmonitors. Es ist also beim physisch kleineren Bildschirm des Tablet-Computers die gleiche Anzahl an Pixeln auf einer wesentlich kleineren Fläche untergebracht. Daraus wird ersichtlich, dass beispielsweise die Angabe einer Schriftgröße von 12 Pixeln auf dem Bildschirm des Tablet-Computers auch deutlich kleiner ausfällt als auf dem Desktopmonitor und die Lesbarkeit ab einer bestimmten Schriftgröße leidet.

Mit Hilfe des Responsive Webdesigns werden nun, abhängig vom Browser, die Maße des Viewports, des Monitors und eventuelle Ausrichtungen

(Hoch- oder Querformat) ausgelesen. Diese Werte können nun wiederum im `<link>`-Element genutzt werden, um für bestimmte Bildschirm- oder Viewportgrößen ein bestimmtes CSS-Stylesheet zu laden. Das nachfolgende Listing zeigt die Einbindung des Stylesheets *styleKlein.css* für Endgeräte mit einer Viewportbreite, die kleiner oder gleich 400 Pixel ist.

```
01 <link type="text/css" rel="stylesheet"
    media="screen and (max-width: 400px)"/>
```

Eine andere Möglichkeit der Einbindung bietet die `@media`-Regel innerhalb eines CSS-Stylesheets, so dass innerhalb eines Stylesheets gleich Angaben für mehrere Größen definiert werden können.

```
01 @media screen and (max-width: 400px) {
02     body {
03         background-color: #ff0000;
04     }
05 }
06
07 @media screen and (min-width: 401px) {
08     body {
09         background-color: #00ff00;
10     }
11 }
12
13 @media screen and (min-width: 1025px) {
14     body {
15         background-color: #0000ff;
16     }
17 }
```

Listing 26: mehrere CSS3 Media Queries innerhalb eines CSS-Stylesheets

Hier wird exemplarisch für Endgeräte mit einer Viewportbreite kleiner oder gleich 400 Pixeln ein roter, größer oder gleich 401 Pixeln ein grüner und größer oder gleich 1025 Pixeln ein blauer Seitenhintergrund angezeigt. Neben `max-` und `min-width` existieren auch deren Entsprechungen für die Viewporthöhe mit `max-` und `min-height`. Eine weitere ausgelesene Eigenschaft neben den Maßen des Viewports sind die Maße

des kompletten Bildschirms. Dafür wird die Anweisung `max-device-width` bzw. `min-device-width` verwendet. Diese wird tatsächlich (derzeit) nur von mobilen Endgeräten unterstützt und von Browsern auf Desktop-Computern ignoriert. Eine weitere für mobile Endgeräte interessante Eigenschaft ist dessen Ausrichtung. Sie kann mit `orientation: landscape` (Querformat) bzw. `orientation: portrait` (Hochformat) abgefragt werden.

Auf diese Weise können entsprechende CSS-Regeln für einzelne Elemente der Website bei bestimmten Auflösungen definiert werden. Die nachstehende Abbildung zeigt beispielhaft, wie die Inhalte einer Website bei verschiedenen Auflösungen (z.B. Desktop-Computer, Tablet-Computer und Smartphones) angeordnet werden könnten.

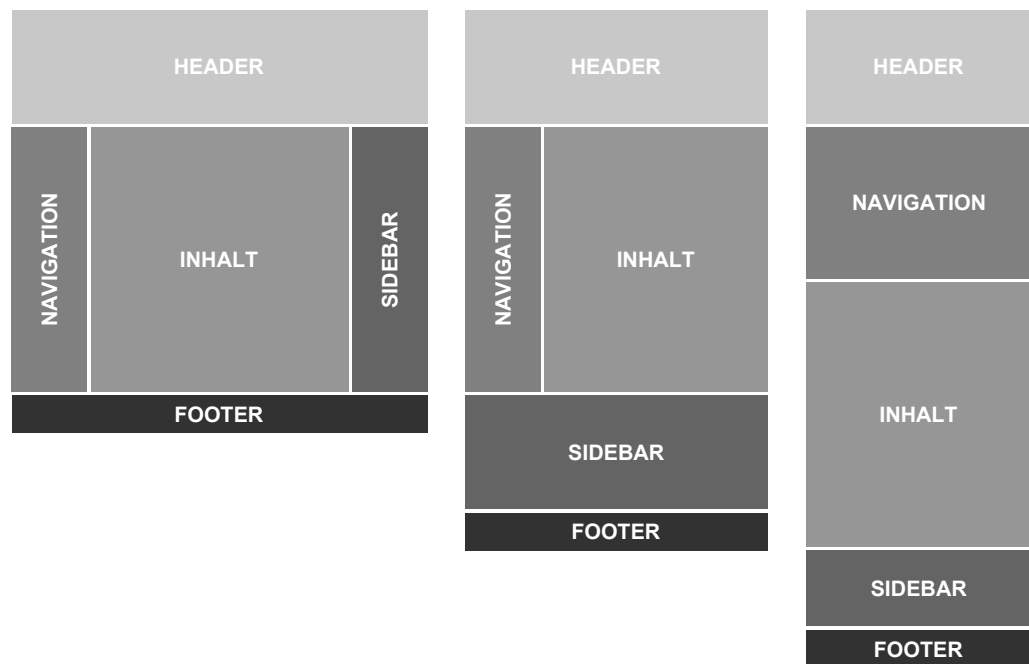


Abbildung 24: Layout bei verschiedenen Auflösungen

Ein wichtiges Merkmal des Responsive Webdesign ist neben dem Anpassen des Layouts auch das Anpassen bestimmter Inhalte. So müssen auf einem Smartphone beispielsweise nicht alle Abbildungen oder Logos angezeigt werden, die auf dem Desktopmonitor angezeigt werden. An dieser Stelle spielt wieder die zielgruppenadäquate Gestaltung eine große Rolle. Benutzer, die eine Website mit einem Smartphone aufrufen,

fen, suchen in der Regel schnell und gezielt nach bestimmten Informationen, während an einem Desktop-Computer oder einem schnellen Laptop durchaus auch längere Texte und multimedial umfangreicher gestaltete Inhalte betrachtet werden. Dieser Hintergrund sollte nicht zuletzt auch unter wirtschaftlichen Gesichtspunkten hinreichende Beachtung finden.

Neben den genannten Techniken existieren noch eine Reihe weiterer, von denen noch insbesondere die *Adaptive Images* Technik erwähnenswert ist (vgl. [WILCOX]). Mit ihr werden Abbildungen für eine Website serverseitig in Verbindung mit JavaScript, PHP und htaccess in der benötigten Größe generiert und erst dann an das Endgerät gesendet. Diese von Matt Wilcox entwickelte Vorgehensweise spielt seine Vorteile insbesondere bei den ohnehin noch recht begrenzten Bandbreiten mobiler Internetanbindungen aus und erspart dem Besucher einer Website die unter Umständen langen Warte- bzw. Ladezeiten bei Grafiken.

3 Praktische Umsetzung

Nachdem im vorherigen Teil dieser Arbeit auf die mit dem Cross-Media-Publishing verbundenen Technologien eingegangen wurde, sollen diese Grundlagen nun am konkreten Beispiel des Benutzerhandbuchs für die Software „WPS-Manager 5.0“ angewendet werden.

3.1 Softwareübersicht

In der Vorbereitungsphase dieser Arbeit wurden verschiedene Softwareprodukte zur Entwicklung des Quelldokuments im DocBook-XML-Format und der XSL-Stylesheets auf ihre Praxistauglichkeit getestet.

Grundsätzlich können zwar sämtliche XML-Dokumente, XSL- oder XSL-FO-Stylesheets in einem Texteditor erstellt und kommandozeilenbasiert transformiert werden. Für die professionelle Entwicklung empfiehlt sich jedoch der Einsatz spezieller XML-Editoren. Bei diesen sind insbesondere Syntax-Highlighting, Code-Vervollständigung, Validierungsfunktionen und zum Teil automatisierbare Funktionen zur Transformation integriert. Nachfolgend sollen einige dieser Editoren aufgeführt und hinsichtlich ihres Funktionsumfangs und besonderer Eigenschaften erläutert werden. Keiner der Editoren ist allein und speziell für die Entwicklung von DocBook-XML-Dokumenten und entsprechenden XSL-Stylesheets ausgelegt. Alle können aber ohne Weiteres für diesen Zweck genutzt werden.

Selbstverständlich stellt die Auswahl keine repräsentative Übersicht aller auf dem Markt erhältlicher Editoren, sondern lediglich eine Auflistung aus subjektiver Sicht besonders geeigneter Produkte dar.

3.1.1 XMLmind XML Editor

Der XMLmind XML Editor von der französischen Firma Pixware, ein kommerzieller XML-Editor, kann sowohl in einer kostenlosen und im Funktionsumfang eingeschränkten Version zur nicht-kommerziellen Nutzung als Personal Edition als auch in der kostenpflichtigen Professional Edition erworben bzw. heruntergeladen werden. Weiterhin existiert der Editor

auch als Java-Applet, so dass er direkt in einem Browser genutzt werden kann. Da diese Software in Java geschrieben wurde, kann sie auf beliebigen Plattformen, auf denen die entsprechende Java-Runtime installiert ist, verwendet werden. Der XMLmind XML Editor bietet allerdings lediglich eine Autorenansicht, so dass keine direkten Änderungen im Quelltext vorgenommen werden können. Dies kann je nach Anwendungsfall als Vor- oder Nachteil empfunden werden. Aufgrund dieses Umstands schied diese Software jedoch als Entwicklungswerkzeug aus, könnte aber durchaus zur reinen Pflege des DocBook-XML-Dokuments genutzt werden (siehe [XMLMIND]).

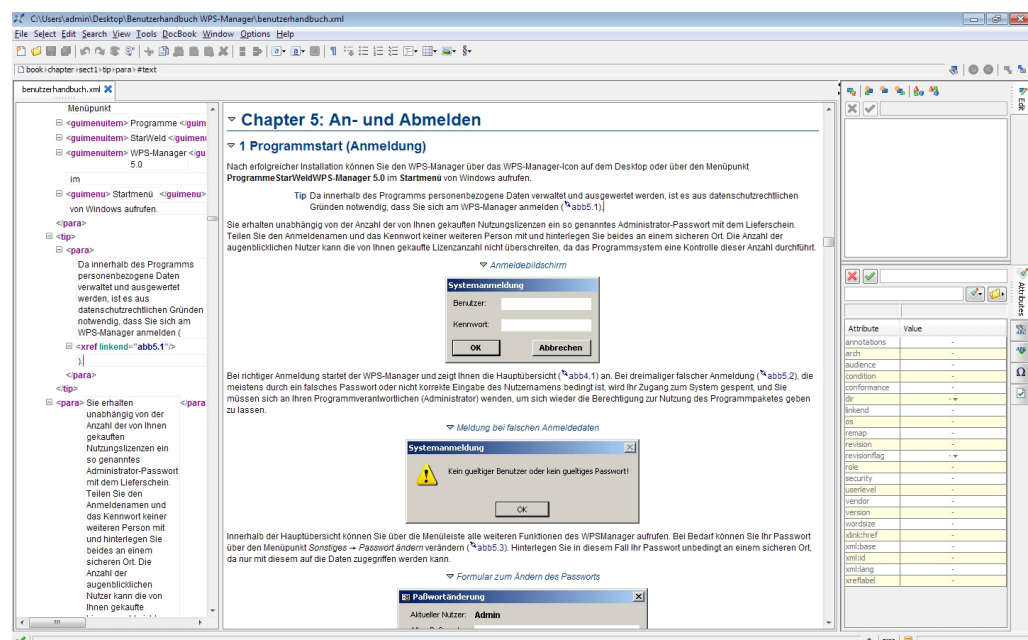


Abbildung 25: DocBook-XML-Dokument in XMLmind XML Editor

3.1.2 Syntext Serna Free

Serna Free vom US-Amerikanischen Hersteller Syntext ist ein Open-Source XML-Editor. Er bietet sowohl eine Autorenansicht, die auf einer XSL-FO-Transformation basiert, und eine rudimentäre Editoransicht. Für kleinere Projekte ist Serna Free durchaus brauchbar, für umfangreiche Projekte sollten jedoch andere Produkte in Betracht gezogen werden. Er zeichnet sich insbesondere durch die flexible Anpassbarkeit seiner Benutzeroberfläche und einer einfachen Bedienung aus. Obwohl dieser

Editor zunächst sehr vielversprechende Ergebnisse lieferte, schied er aufgrund verschiedener Einschränkungen hinsichtlich der Anpassbarkeit der Autorenansicht, dem Fehlen eines umfangreichen Quelltexteditors und dem Fehlen einer DocBook 5 Unterstützung ebenfalls als Entwicklungswerkzeug aus (siehe [SERNA FREE]).

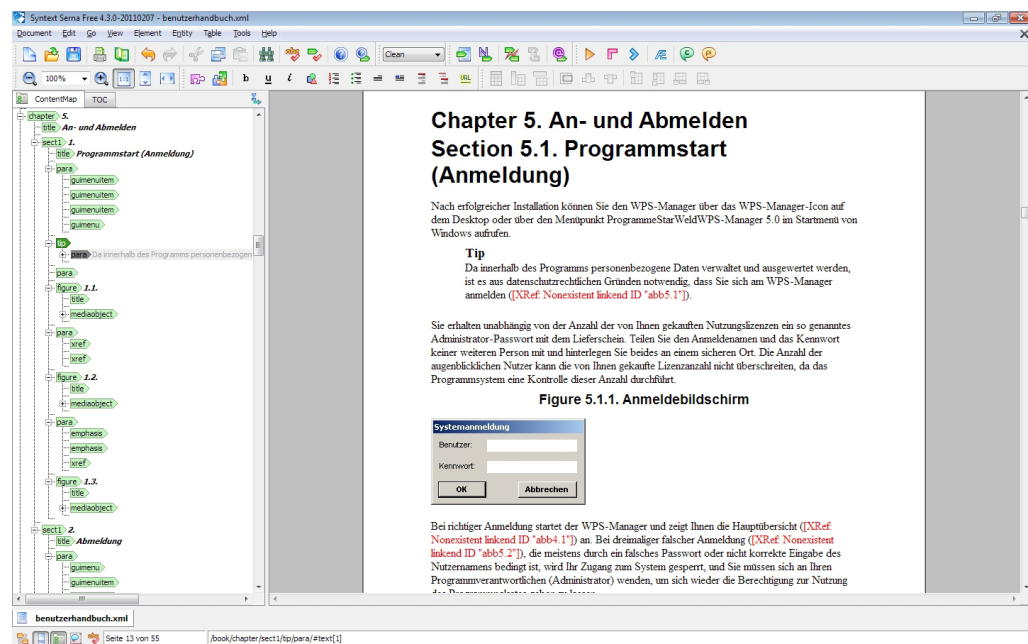


Abbildung 26: DocBook-XML-Dokument in Serna Free

3.1.3 Oxygen XML Editor

Der Oxygen XML Editor ist ein in Java geschriebener – und damit ebenfalls plattformunabhängiger – kommerzieller XML Editor der Firma SyncRO SOFT Ltd. Er ist in den Versionen XML Developer, XML Author und XML Editor erhältlich, wobei letzterer die beiden ersten Versionen enthält. Insbesondere die Möglichkeit zwischen den Ansichten Text (Quellcode), Raster (Inhalt als verschachtelte Tabelle) und Autor (per CSS formatierter Vorschaumodus) umzuschalten und die Möglichkeit, verschiedene Transformationsszenarien zu speichern, zeichnen dieses Produkt aus (siehe [OXYGEN]).

Dieses Produkt wurde in einer Academic License ausgiebig getestet und aufgrund seiner enormen Fähigkeiten für die Entwicklung der XSL-

Stylesheets des erstellten Benutzerhandbuchs verwendet. Auf der Website des Herstellers werden neben einem Forum auch Anleitungsvideos in englischer Sprache angeboten, die den Einstieg im Umgang mit der Software deutlich erleichtern und einen guten Überblick über den Funktionsumfang bieten.

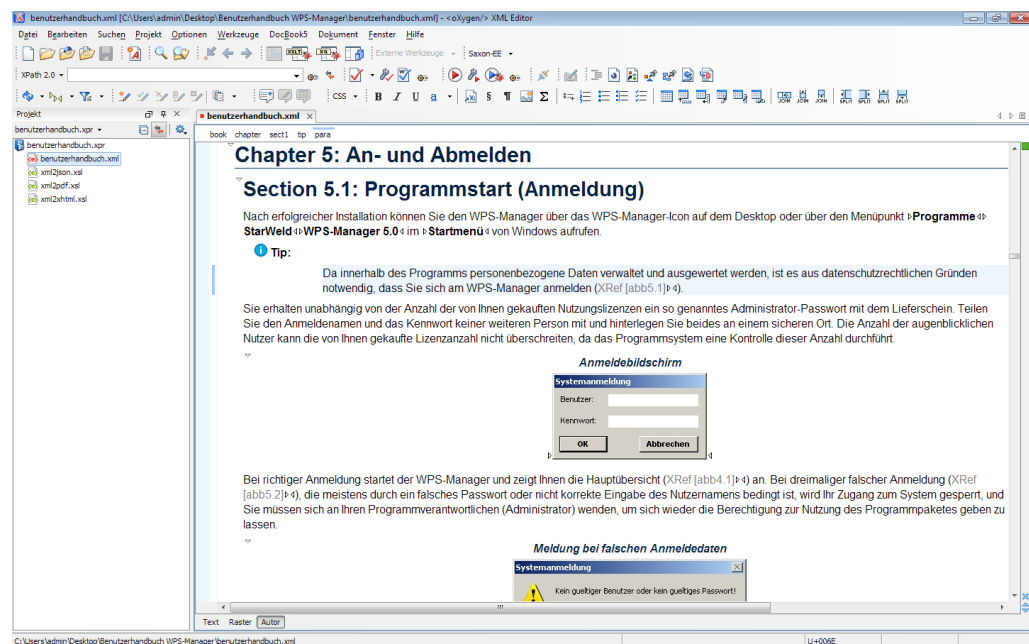


Abbildung 27: DocBook-XML-Dokument im Oxygen XML Editor

3.1.4 XML Copy Editor

Der kostenlose XML Copy Editor vom Entwickler Gerald Schmidt bietet lediglich eine Editoransicht für den reinen Quellcode. Er kann jedoch auf einfache Weise die Wohlgeformtheit und Gültigkeit eines XML-Dokuments prüfen und zeichnet sich insbesondere durch seine Schnelligkeit aus. Auch dieser Editor ist eher für kleine Projekte oder für schnelle Änderungen an XML-Dokumenten geeignet. Er existiert in einer Windows- und einer Linuxversion (siehe [XMLCOPY]).

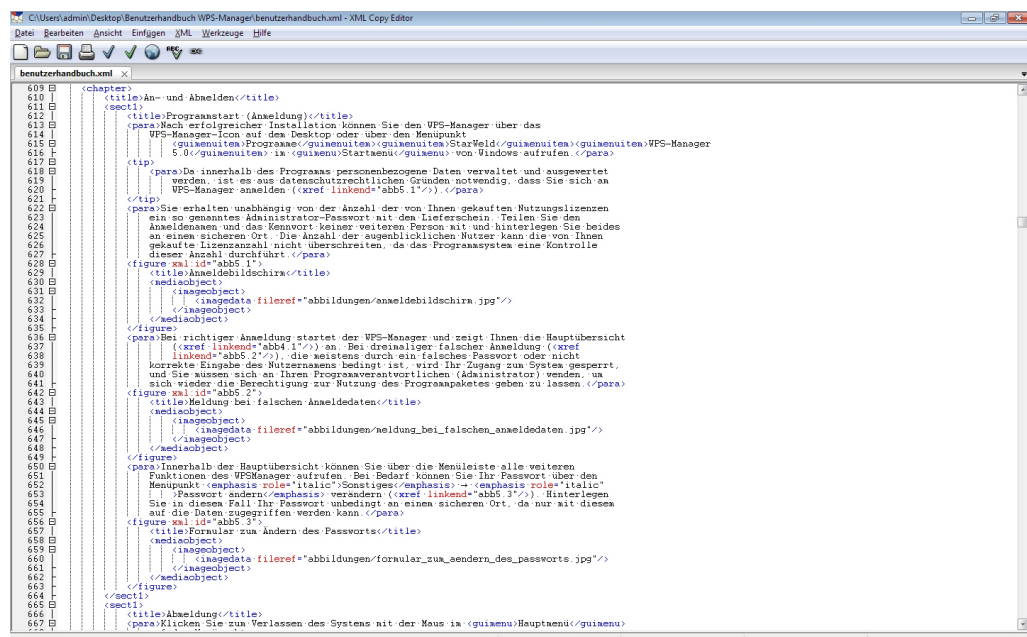


Abbildung 28: XSL-Stylesheet im XML Copy Editor

3.1.5 Webocton Scriptly

Webocton Scriptly ist kein reiner XML-Editor, kann aber auch für diese Zwecke verwendet werden. Er wird als Freeware-Projekt für Windows von Benedikt Loepp entwickelt und ist insbesondere für die Entwicklung von (X)HTML, CSS, PHP und JavaScript geeignet. Der in Python geschriebene Editor bietet unter anderem einen FTP-Client, eine umfangreiche Projektverwaltung, einen MySQL-Assistenten und einen Dateivergleicher. Er kann aufgrund seines Funktionsumfangs ohne weiteres mit großen kommerziellen Produkten mithalten und stellt ein universelles Werkzeug zur Webentwicklung dar (siehe [SCRIPTLY]).

Diese Software wurde zur Entwicklung der CSS-Stylesheets und der JavaScript-Funktionen für die XHTML-Ausgabe verwendet.

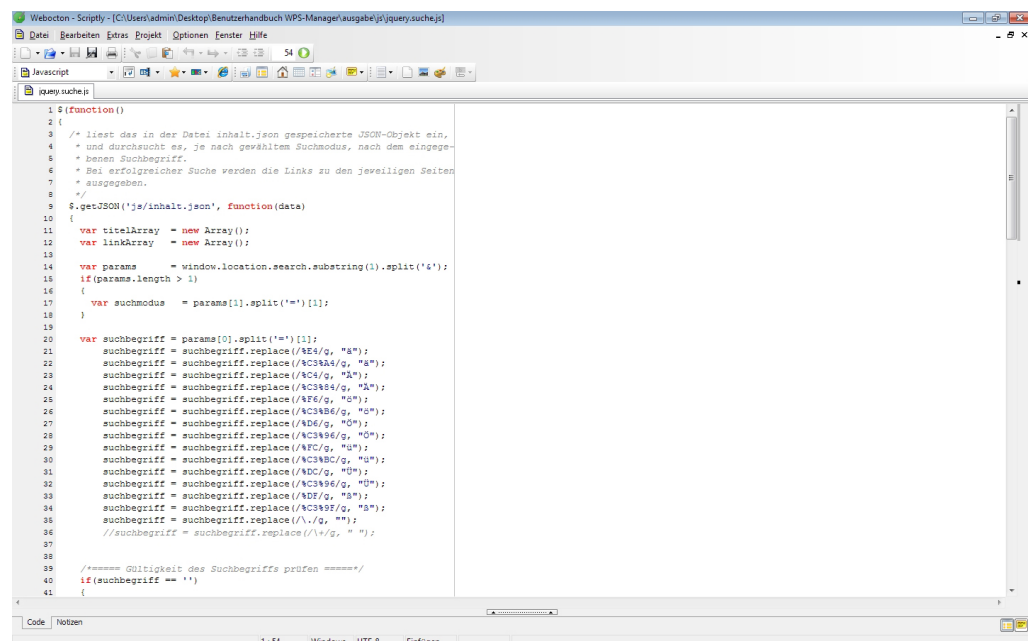


Abbildung 29: JavaScript-Dokument in Webocton Scriptly

3.2 Verwendete Entwicklungsumgebung

Nachdem im vorherigen Abschnitt verschiedene Softwareprodukte aufgelistet wurden, die zur Erstellung und Bearbeitung von DocBook-XML-Dokumenten und XSL-Stylesheets verwendet werden können, soll in diesem Abschnitt die tatsächlich zur Entwicklung des Benutzerhandbuchs genutzte Entwicklungsumgebung erläutert werden. Dabei liegt der Fokus der Betrachtungen auf dem XML-Editor Oxygen XML Editor, welcher aufgrund seines Funktionsumfangs für den Großteil der Arbeitsschritte das geeignetste Werkzeug darstellt.

Die zur Transformation des XML-Quelldokuments in die Zielformate XHTML und PDF genutzten Prozessoren Saxon HE (Java-Version) und Apache FOP sind in Java geschriebene Programme und benötigen für ihre Funktion eine auf dem Rechner installierte Java-Runtime. Apache FOP benötigt in seiner aktuellsten Version 1.0 die Java-Runtime ab Version 1.4, Saxon HE benötigt die Java-Runtime ab Version 1.5.

Im Installationspaket von Oxygen XML Editor in der derzeit aktuellen Version 13.1 sind die entsprechenden XSLT- bzw. FO-Prozessoren inklusive

der Java-Runtime 1.6 enthalten und brauchen somit nicht separat installiert zu werden.

3.2.1 Silbentrennung in Apache FOP

Aus lizenzrechtlichen Gründen ist das Modul zur Silbentrennung bzw. die Silbentrennungsmuster für PDF-Dokumente weder im Oxygen XML Editor noch in Apache FOP direkt integriert. Die entsprechend benötigte Datei *fop-hyph.jar* kann jedoch von der OFFO-Projektwebsite (siehe [OFFO]) kostenlos heruntergeladen und anschließend in den Unterordner *lib* innerhalb des Installationsordners von Oxygen XML Editor kopiert werden, um diese Funktionalität nutzen zu können.

3.2.2 Darstellungsfehler in Apache FOP

An dieser Stelle soll auf ein Bug in Apache FOP hingewiesen werden. Wenn die Silbentrennung aktiviert ist (`hyphenate="true"`), werden Rahmen und Innenabstände von `<fo:inline>`-Elementen nicht korrekt angezeigt, wie die nachstehenden Abbildungen zeigen.

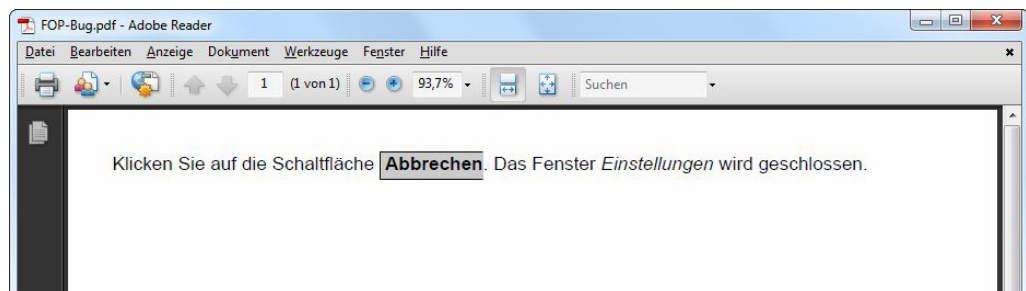


Abbildung 30: Darstellungsfehler bei aktivierter Silbentrennung

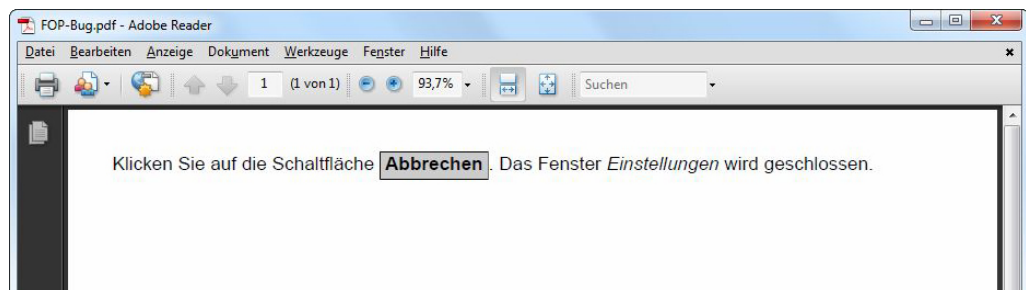


Abbildung 31: kein Darstellungsfehler bei deaktivierter Silbentrennung

Dieser Fehler ist als *Bug 49870* bekannt (siehe [APACHE 1]) und soll laut Entwickler in *Revision 1050342* behoben sein (siehe [APACHE 2]). Trotz dessen ist dieser Bug nach wie vor in der zum Zeitpunkt der Erstellung dieser Arbeit aktuellsten Version 13.1 von Oxygen XML Editor vorhanden.

3.2.3 Benutzerdefinierte Autorenansicht in Oxygen XML Editor

Ein besonders hervorzuhebendes Merkmal vom Oxygen XML Editor ist die Anpassbarkeit der Autorenansicht des Programms. Das Aussehen dieser Autorenansicht, welche einer Textverarbeitungssoftware ähnelt, kann über CSS-Stylesheets formatiert werden. Damit kann die Vorschau den eigenen Bedürfnissen bzw. dem Aussehen des Zielformats in gewissem Maße angepasst werden.

Der einfachste Weg besteht in der Abänderung der bestehenden CSS-Stylesheets. Diese sind in den jeweiligen Frameworkordnern im Unterordner `/frameworks` des Installationsordners von Oxygen XML Editor gespeichert. Für das DocBook-Framework wäre der Pfad beispielsweise `C:/Program Files (x86)/Oxygen XML Editor 13/frameworks/docbook/css`. Die dort zu findende Datei *docbook.css* enthält den Großteil der Formatierungsanweisungen für die Autorenansicht eines DocBook-XML-Dokuments.

Soll nun beispielsweise die standardmäßig verwendete Schriftart Times New Roman durch Arial ersetzt, die englischen Bezeichnungen für Chapter und Section in der Autorenansicht in ihre deutschen Entsprechungen umgewandelt, und Abbildungen sowie deren Beschriftungen linksbündig statt standardmäßig zentriert angeordnet werden, müssen in der Datei *docbook.css* folgende Änderungen vorgenommen werden.

In Zeile 11 einfügen:

```
01      * {  
02          font-family: Arial, sans-serif;
```

```
03 }
```

In Zeile 126 ersetzen von:

```
01 content: "Chapter " counter(chapter_count, deci-  
mal) ": ";
```

durch

```
01 content: "Kapitel " counter(chapter_count, deci-  
mal) ": ";
```

Alle Vorkommen (10 mal) von:

```
01 content: "Section "
```

ersetzen durch

```
01 content: "Abschnitt "
```

In Zeile 1948 und 1904 ersetzen von

```
01 text-align: center;
```

durch

```
01 text-align: left;
```

Die beiden nachstehenden Abbildungen zeigen einen Vergleich der standardmäßigen Autorenansicht und der Autorensicht nach den Änderungen an der Datei *docbook.css*.

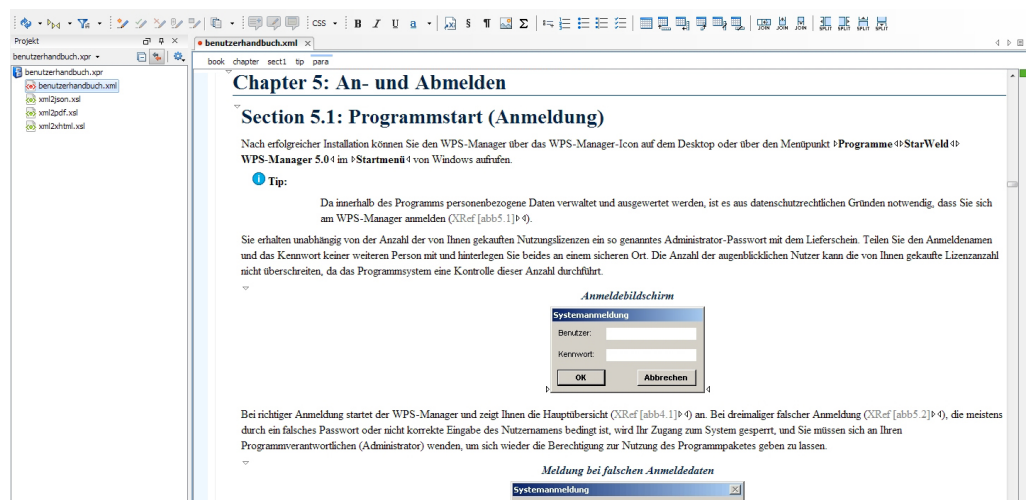


Abbildung 32: unbearbeitete Autorenansicht in Oxygen XML Editor

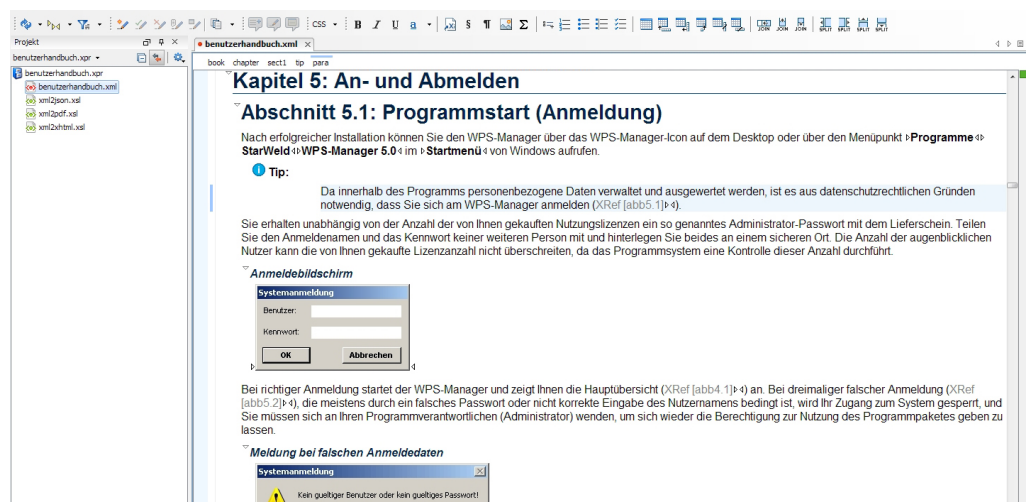


Abbildung 33: bearbeitete Autorenansicht in Oxygen XML Editor

Mit einigen CSS-Kenntnissen können hier vielfältige Änderungen vorgenommen werden und das Aussehen der Autorenansicht den eigenen Bedürfnissen angepasst werden. Idealerweise sollte dabei vor dem Bearbeiten eine Sicherheitskopie der Dateien erstellt werden. Die direkte Bearbeitung der Dateien im Installationsverzeichnis benötigt Administratorrechte.

Die Programmhilfe vom Oxygen XML Editor bietet weitere Hinweise zur Anpassung der Autorenansicht sowie zu Erstellung und Austausch eigener Dokumenttyp-Zuordnungen mit entsprechend formatierten Auto-

renansichten bis hin zur Java-basierten Entwicklung benutzerdefinierter Schaltflächen innerhalb der Programmoberfläche.

3.3 XSL-Stylesheets

Wie in *Abschnitt 2.4* erwähnt, wurden für die Entwicklung des Benutzerhandbuchs eigene XSL-Stylesheets entwickelt. Es wurden insgesamt drei Stylesheets für die Transformation des DocBook-XML-Quelldokuments in die jeweiligen Zielformate erstellt. Zum ersten das Stylesheet *xml2xhtml.xsl*, welches mehrere XHTML-Dokumente erzeugt. Zum zweiten das Stylesheet *xml2pdf.xsl*, was zunächst ein XSL-FO-Dokument, in einem weiteren Verarbeitungsschritt dann ein PDF-Dokument erzeugt. Zum dritten das XSL-Stylesheet *xml2json.xsl*, welches ein JSON-Objekt erzeugt, welches die reinen Textinhalte, Titel und Dateinamen aller XHTML-Dokumente des Benutzerhandbuchs für die Suchfunktion (siehe *Abschnitt 3.9.9*) enthält.

Diese XSL-Stylesheets können der dieser Arbeit beiliegenden CD-ROM entnommen werden.

3.4 Zielformate

Entsprechend der Zielsetzung wurde das Benutzerhandbuch in zwei verschiedenen Zielformaten ausgegeben. Zum einen in Form einer Sammlung von miteinander verlinkten XHTML-Dokumenten als Online-Dokumentation und als PDF-Dokument, welches bei Bedarf ausgedruckt oder ebenfalls am Bildschirm gelesen werden kann.

Selbstverständlich wäre es durchaus möglich, ein einziges (X)HTML-Dokument zu erzeugen, welches alle Kapitel und Abschnitte enthält und über die entsprechende Funktion des Browsers ausgedruckt werden kann. Allerdings sollte eine hochwertige Druckausgabe ermöglicht werden, wie sie nur ein speziell für diesen Zweck geschaffenes Format wie das PDF-Format liefert.

3.4.1 XHTML

XHTML (Extensible Hypertext Markup Language) ist eine weitere, im Gegensatz zu HTML auf XML basierende, Auszeichnungssprache. Sie liegt seit dem 26. Januar 2000 als Empfehlung des W3C vor ([W3C 12]), welche am 01. August 2002 in einer überarbeiteten Version erneut veröffentlicht wurde ([W3C 13]). Wie der Untertitel der Empfehlung:

A Reformulation of HTML 4 in XML 1.0

ist XHTML eine Neuformulierung von HTML 4. Während die Auszeichnungssprache HTML auf die seit 1986 existierende Metasprache SGML basiert, welche zwar sehr ausgereift aber auch sehr komplex ist, sollte mit XHTML ein auf XML basierendes, gleichwertiges Pendant geschaffen werden.

3.4.2 PDF

Das PDF-Format (Portable Document Format) ist ein weitverbreitetes Format zum plattformunabhängigen Austausch von Dokumenten. Es wird seit 1993 von amerikanischen Softwareunternehmen Adobe Systems entwickelt und bietet den großen Vorteil, dass sämtliche Medien, wie etwa den eigentlichen Text, die Schriftarten, in denen der Text angezeigt werden soll, Bilder oder sogar 3D-Animationen und Videos, in einer einzigen Datei vorliegen. Im Gegensatz zu (X)HTML kann das Aussehen eines PDF-Dokuments nicht durch äußere Umstände beeinflusst werden, so dass es immer und auf jeder Plattform der vom Ersteller festgelegten Form entspricht.

3.5 Informationsstruktur

Neben der Festlegung der eigentlich zu vermittelnden Informationen ist es ebenso von großer Bedeutung, diese Informationen in einer sinnvollen Form zu strukturieren. So gilt es zunächst die zu vermittelnden Informationen bzw. Handlungsanweisungen entsprechend ihrer Reihenfolge zu strukturieren. Es wäre beispielsweise wenig zielführend, die Suche nach

bestimmten Schweißanweisungen zu erläutern, bevor erklärt wurde, wie Schweißanweisungen erstellt und gespeichert werden.

Weiterhin ist bei der Strukturierung der Informationen zu beachten, dass bestimmte Informationen vor anderen Informationen bekannt sein müssen. Ein konkretes Beispiel, zumindest bei älteren Versionen der Software, ist hier etwa die Wahl des Betriebssystems. Bevor die Software installiert und verwendet werden kann, muss sichergestellt sein, dass ein passendes Betriebssystem installiert ist.

Zusätzlich gilt es, die Informationseinheiten so zu strukturieren, dass immer wieder benötigte Informationen möglichst zusammenhängend und schnell auffindbar sind. An dieser Stelle spielt die enge Verknüpfung der sprachlichen und der gestalterischen Umsetzung eine besonders wichtige Rolle.

Nicht zuletzt muss bei der Informationsstrukturierung auch die Relevanz einzelner Informationen beachtet werden. So wird bei der Erstbenutzung der Software wohl zunächst die Benutzeroberfläche und die Menüstruktur von Interesse sein, wogegen der Benutzer vermutlich erst zu einem späteren Zeitpunkt beispielsweise Grafiken in Schweißanweisungen einfügen will.

Um einer sinnvollen Unterteilung des Benutzerhandbuchs Genüge zu tragen, wurde die nachstehende inhaltliche Strukturierung gewählt.

- 1 Leistungsmerkmale**
- 2 Systemanforderungen**
 - 2.1 Hardware**
 - 2.2 Software**
- 3 Installation**
 - 3.1 Netzwerkinstallation**
 - 3.2 Einzelplatzinstallation**
- 4 Erste Schritte nach der Installation**
 - 4.1 Standarddrucker auswählen**
 - 4.2 Datenbasis anpassen**
 - 4.3 Programmeinstellungen anpassen**
- 5 An- und Abmeldevorgang**
- 6 Benutzeroberfläche**
 - 6.1 Symbolleiste**

- 6.2 Listen**
 - 6.2.1 Sortieren von Listeneinträgen
 - 6.2.2 Filtern von Listeneinträgen
 - 6.2.3 Benutzerdefinierte Filter
 - 6.3 Formulare**
 - 6.4 Anpassen der Benutzeroberfläche**
- 7 Nutzerverwaltung**
 - 7.1 Hinzufügen von Nutzern
 - 7.2 Löschen von Nutzern
 - 7.3 Bearbeiten von Nutzern
- 8 Schweißerverwaltung**
 - 8.1 Schweißerstammdaten eingeben, speichern, ändern, drucken und löschen
 - 8.2 Zuordnung einer Schweißprüfung zu einem Schweißer
- 9 Schweißprüfungsverwaltung**
 - 9.1 Schweißprüfungen eingeben, speichern, ändern, drucken und löschen
 - 9.2 Schweißprüfungen verlängern
 - 9.3 Schweißprüfungen Menü
 - 9.4 Schweißprüfungen auflisten
- 10 Arbeitsprobenverwaltung**
- 11 Schweißanweisungsverwaltung**
 - 11.1 Schweißanweisungen auflisten
 - 11.2 Schweißanweisungen erstellen, speichern, ändern, drucken und löschen
 - 11.3 Einfügen von Grafiken in eine Schweißanweisung
 - 11.3.1 Einführung zum Einfügen von Grafiken
 - 11.3.2 Einfügen von Grafiken aus einer Datenbank
 - 11.3.3 Einfügen von Grafiken aus einer Datei
 - 11.4 Schweißanweisungen suchen
 - 11.5 Schweißanweisungen drucken
- 12 Schweißverfahrensprüfungsverwaltung**
- 13 Datenbasisverwaltung**
- 14 Tipps zur Bedienung**
 - 14.1 Datensicherung
 - 14.2 Programmstruktur
 - 14.3 Häufig gestellte Fragen

3.6 Sprachliche Gestaltung

Wie bereits in *Abschnitt 2.11* erläutert, gilt es bei der Gestaltung von Anleitungen bestimmte Regeln hinsichtlich der sprachlichen Gestaltung zu berücksichtigen.

3.6.1 Makrostruktur

Die Makrostruktur, also die semantische Inhaltsstrukturierung, unterscheidet sich bei der XHTML-Ausgabe leicht von der PDF-Ausgabe. Der Grund liegt in der potentiell nichtlinearen Nutzbarkeit der XHTML-Ausgabe. Hier kann der Benutzer mittels diverser Verlinkungen schnell zu anderen Abschnitten springen, ohne zwischenliegende Informationseinheiten zu registrieren. Deshalb wird bei diesem Ausgabeformat zusätzlich in Form einer Unternavigation auf eventuell vorhandene tieferliegende Abschnitte hingewiesen.

Grundsätzlich sind beide Ausgaben jedoch makrostrukturell gleich aufgebaut, so dass sich eine typische Struktur ergibt.

```
1 chapter
  1.1 sect1
  1.2 sect1
    1.2.1 sect2
  1.3 sect1
2 chapter
  2.1 sect1
```

Diese Strukturierung ergibt sich aus dem Schema des oben beschriebenen DocBook-Formats.

Jedes Kapitel `<chapter>` und jeder Abschnitt `<sect1>` bis max. `<sect5>` wird durch eine Überschrift `<title>` eingeleitet, wobei diese mit einer in beiden Zielformaten identischen Nummerierung versehen wird.

Grundsätzlich können beliebig viele Kapitel mit beliebig vielen Abschnitten versehen werden, wobei diese Abschnitte wiederum weitere Unterabschnitte enthalten können. Aus Gründen der Übersichtlichkeit und zur einfacheren Erzeugung des Inhaltsverzeichnisses für die XHTML-Ausgabe wurde im XSL-Stylesheet eine maximale Verschachtelungstiefe bis zu `<sect4>`, also bis zu fünf Ebenen, definiert.

Die XHTML-Ausgabe enthält, wie weiter oben erwähnt, zusätzlich eine Unternavigation unter der Kapitel-/Abschnittsüberschrift, wenn sich unterhalb der Hierarchieebene weitere Abschnitte befinden. Weiterhin ent-

hält sie eine so genannte Breadcrumb-Navigation, welche dem Benutzer den hierarchischen Pfad zum aktuellen Abschnitt visualisiert.

Kapitel und Abschnitte können ihrerseits Absätze, Abbildungen, Tabellen und Listen enthalten. Abbildungen und Tabellen werden dabei mit nummerierten Bildunterschriften versehen.

3.6.2 Mikrostruktur

Die Mikrostruktur folgt im Allgemeinen dem erwähnten Schema von Handlungsanweisung → Handlungsresultat. Dabei wird die Handlungsanweisung stilistisch einfach und aktiv-positiv formuliert (*Bsp.: „Klicken Sie auf die Schaltfläche...“, „Wählen Sie im Hauptmenü...“, „Markieren Sie die Zeile...“*).

Wird auf bestimmte Schaltflächen oder Symbole der Benutzeroberfläche hingewiesen, werden diese inzeilig eingebunden (Symbole) oder speziell formatiert (Schaltflächen), wobei letztere nicht umgebrochen werden.

Zur Hervorhebung bestimmter wichtiger Textpassagen werden diese farbig hinterlegt und mit einem entsprechenden Symbol versehen. Innerhalb des XML-Quelldokuments werden dafür das `<tip>`-Element für Hinweise und das `<note>`-Element für Bemerkungen, die für die korrekte Funktionsweise der Software von Nöten ist, genutzt. Die nachstehende Tabelle zeigt die entsprechenden Symbole und Farben für diese Hervorhebungen.

	DocBook-Element	Symbol	Farbe	
			XHTML	PDF
Hinweis	<code><tip></code>		#A5CBF0	cmyk(0.3, 0.1, 0, 0)
Achtung	<code><note></code>		#FFAC9B	cmyk(0, 0.35, 0.35, 0)

Tabelle 6: Symbole und Farben für Hervorhebungen

3.7 Layout

Das Layout beider Zielformate (XHTML und PDF) orientiert sich am Corporate Design des Unternehmens. Dazu zählen neben den Logos des Unternehmens auch im CMYK- und RGB-Farbmodus vorgegebene Farben und die Verwendung der Myriad-Pro-Schriftfamilie. Dabei unterscheidet sich das Layout der XHTML-Ausgabe prinzipbedingt von dem der PDF-Ausgabe.

3.7.1 PDF-Ausgabe

Die PDF-Ausgabe entspricht dem DIN A4-Format mit einer Seitenlänge von 210 mm x 297 mm und wurde mit einem einseitigen Satzspiegel entwickelt.

Der Satzspiegel ist der Teil einer Druckseite, welcher von den Textspalten und Abbildungen eingenommen wird. Die Ränder um den Satzspiegel werden dabei als Stege (Kopfstege, Fußstege, Bundstege, Außenstege) bezeichnet. Während Kolumnentitel, also beispielsweise die auf jeder Seite in der Kopfzeile ausgegebenen Kapitel- und Abschnittstitel zum Satzspiegel gerechnet werden, würden Kopf- und Fußzeilen, die lediglich eine Seitennummerierung enthalten, nicht zu diesem gerechnet werden.

Bei den Inhaltsseiten des Benutzerhandbuchs wurde auf eine farbige Hintergrundgestaltung und unnötige Layoutelemente verzichtet. Diese Entscheidung basiert auf ökonomischen und in gewisser Hinsicht auch ökologischen Gründen, da sich Benutzer das Benutzerhandbuch bei Bedarf ausdrucken können und dies bei einem Handbuch diesen Umfangs relativ viel Tinte verbrauchen würden.

Die nachstehenden Abbildungen zeigen schematisch das Layout des Deckblatts sowie der Rückseite und der Inhaltsseiten. In den jeweils darunterstehenden Tabellen sind die Maße und Positionen der einzelnen Layoutelemente definiert.

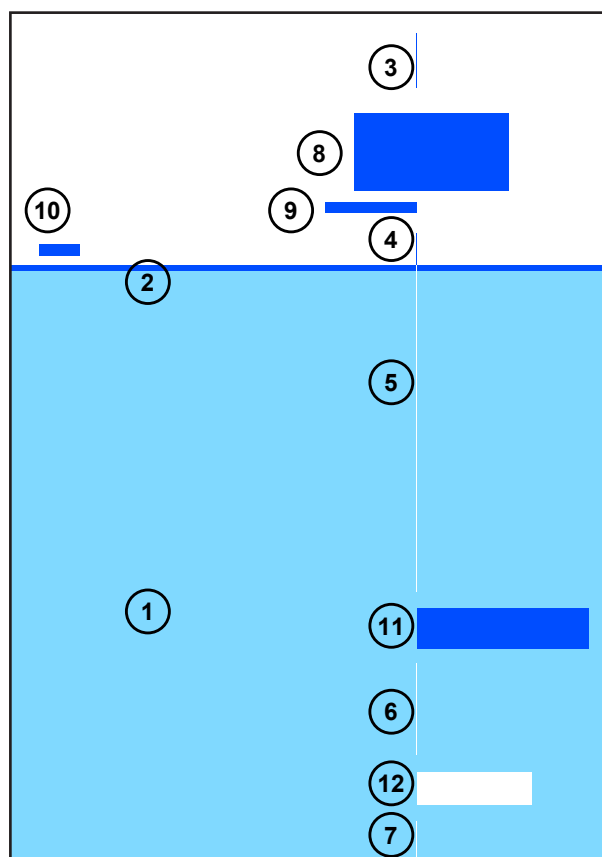


Abbildung 34: Layout des Deckblatts

Nr.	X in mm	Y in mm	Breite in mm	Höhe in mm
1	0	88	210	209
2	0	88	210	2
3	141,75	7	0,25	19
4	141,75	77	0,25	11
5	141,75	88	0,25	114
6	141,75	227	0,25	32
7	141,75	282	0,25	15
8	120	35	54	27,21
9	110	66	32	3,72
10	10	81	14	3,84
11	142	208	auto	auto
12	142	265	40	11,4

Tabelle 7: Positionierung der Layoutelemente des Deckblatts

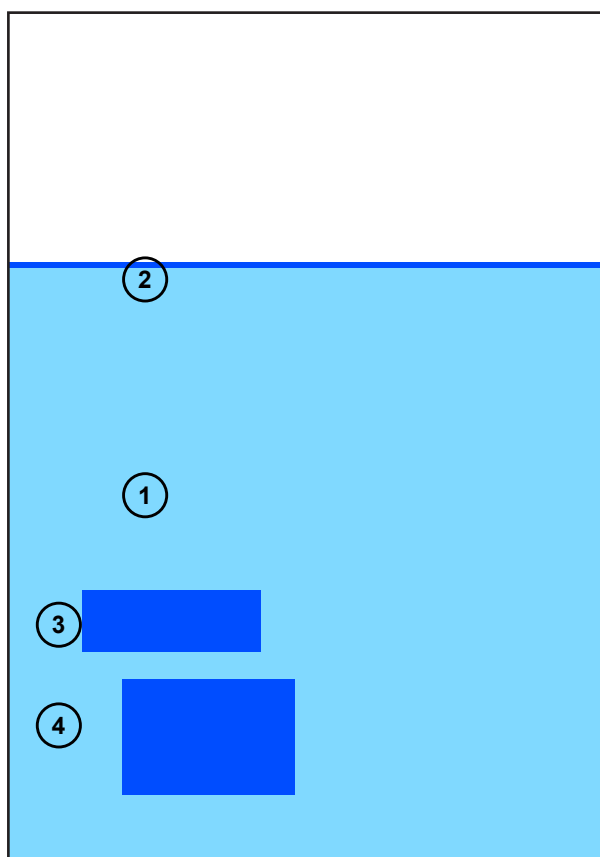


Abbildung 35: Layout der Rückseite

Nr.	X in mm	Y in mm	Breite in mm	Höhe in mm
1	0	88	210	209
2	0	88	210	2
3	26	202	62	21,28
4	40	233	auto	auto

Tabelle 8: Positionierung der Layoutelemente der Rückseite

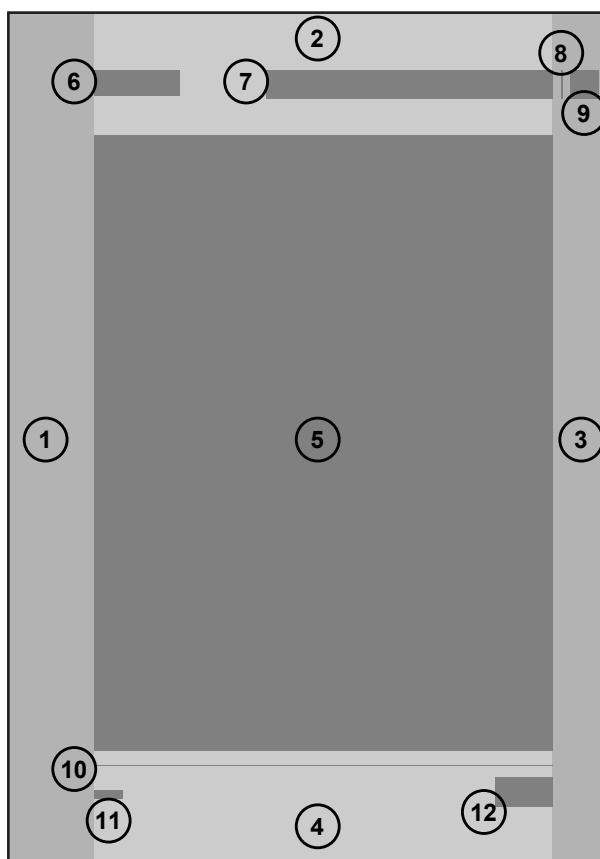


Abbildung 36: Layout der Inhaltsseiten

Nr.	X in mm	Y in mm	Breite in mm	Höhe in mm
1	0	0	30	297
2	0	0	210	42,6
3	190	0	20	297
4	0	257,6	210	39,4
5	30	42,6	160	215
6	30	20	30	8,95
7	90	20	100	auto
8	193	21	0,5	10
9	196	20	10	10
10	30	263	160	0,25
11	30	271,35	10	3
12	170	267	20	10,26

Tabelle 9: Positionierung der Layoutelemente der Inhaltsseiten

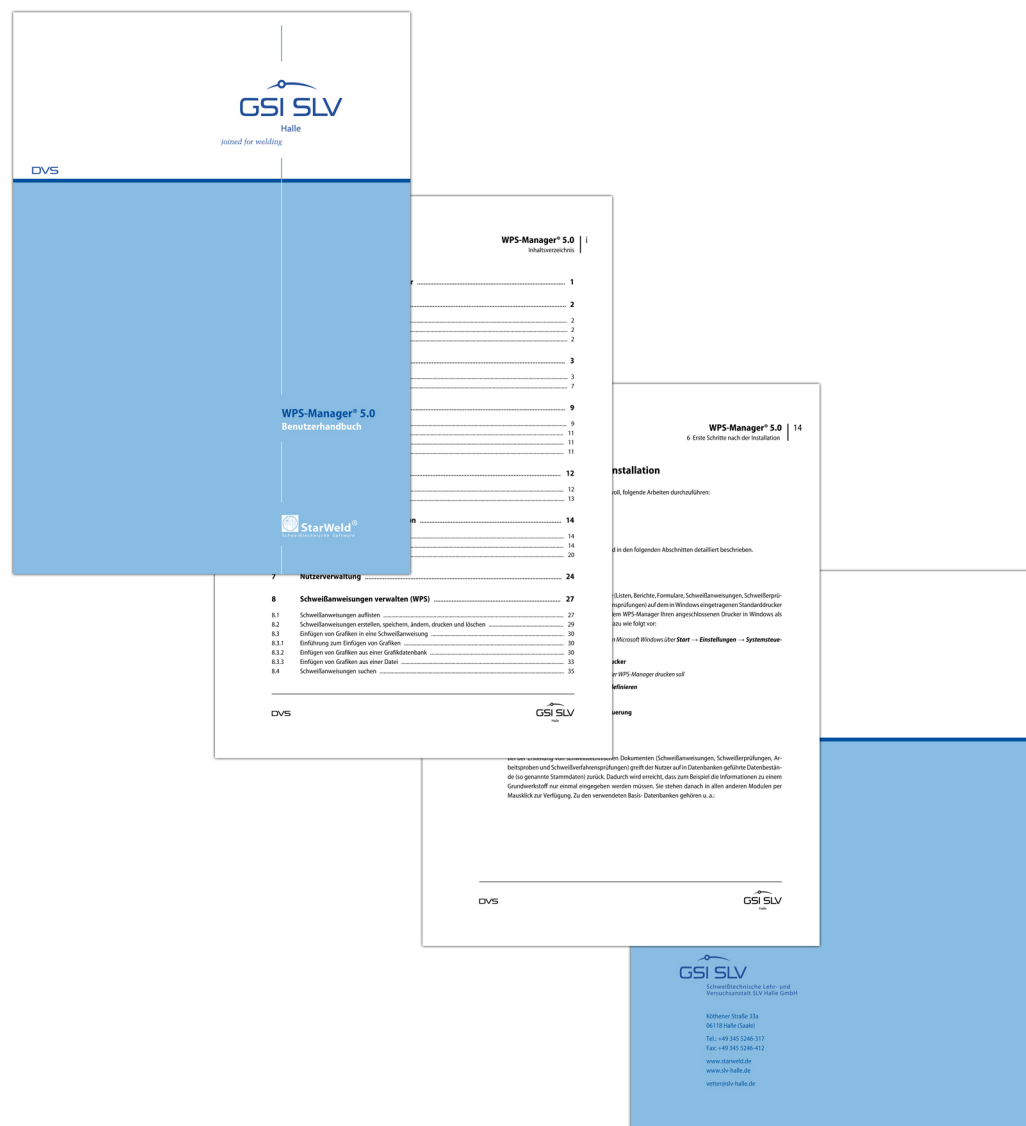


Abbildung 37: Beispielseiten des Zielformats PDF

3.7.2 XHTML-Ausgabe

Die XHTML-Ausgabe wurde zweispaltig mit Kopfzeile entwickelt, wobei ein Großteil der Elemente mit fixen Maßen versehen wurde, auch wenn die Verwendung fixer Maße in gewissem Sinne dem Grundgedanken des Responsive Webdesign widerspricht (hier sollte vorzugsweise mit prozentualen Angaben gearbeitet werden). Die Anpassung an verschiedenen große Endgeräte wurde durch die im *Abschnitt 2.13* beschriebenen CSS3 Media Queries und entsprechende Regeln innerhalb des CSS-Styles-

heets realisiert. Dabei wurden Regeln für Endgeräte mit Bildschirmbreiten bis 768 px (z.B. Smartphones und Tablet-Computer) und Endgeräte mit Bildschirmbreiten über 768 px (z.B. Laptops und Desktop-Computer) definiert. Weitere Untergliederungen wären zwar grundsätzlich möglich, erschienen aber aufgrund der Zielgruppe und des zu kommunizierenden Inhalts des Benutzerhandbuchs als unnötig. Verschiedene Elemente, wie die Logos im Header und der Download-Link des Handbuchs im PDF-Format werden bei verkleinerter Ansicht nicht mehr angezeigt. Ebenso wird das Layout bei verkleinerter Ansicht vertikal linearisiert, so dass die Navigation nicht mehr links neben dem Inhalt platziert ist, sondern über diesem angezeigt wird. Somit entfällt bei Endgeräten mit kleinerer Bildschirmgröße das lästige horizontale Scrollen. Die Prinziplayouts beider Varianten sind nachfolgend dargestellt.

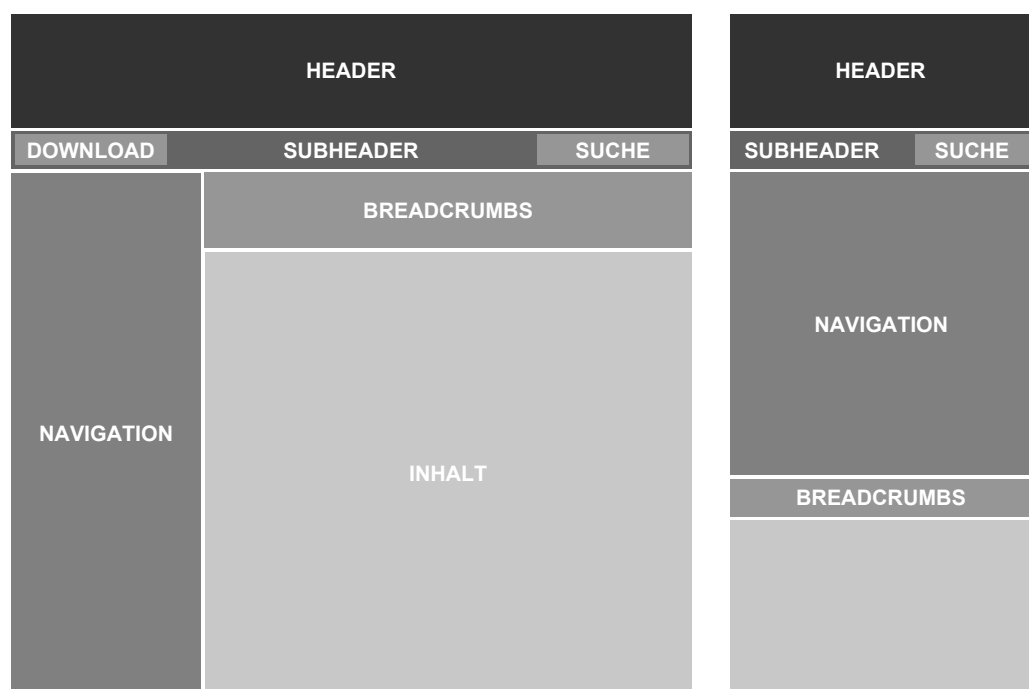


Abbildung 38: Layout der XHTML-Ausgabe

Die verwendeten Farben der XHTML-Ausgabe entsprechen den im Corporate Design festgelegten RGB-Angaben und wurden in entsprechende hexadezimale Werte umgerechnet. Lediglich die Schriftart entspricht nicht der Hausschrift des Unternehmens. Die Hausschrift Myriad-Pro könnte zwar mit der `@font-face`-Regel ohne weiteres im CSS-Stylesheet in-

tegriert werden, dies wurde aber aufgrund urheberrechtlicher Bedenken nicht realisiert.

Die nachstehende Abbildung zeigt die XHTML-Ausgabe in Bildschirmauflösungen, die denen eines Desktop-Computers und eines Smartphones entsprechen.

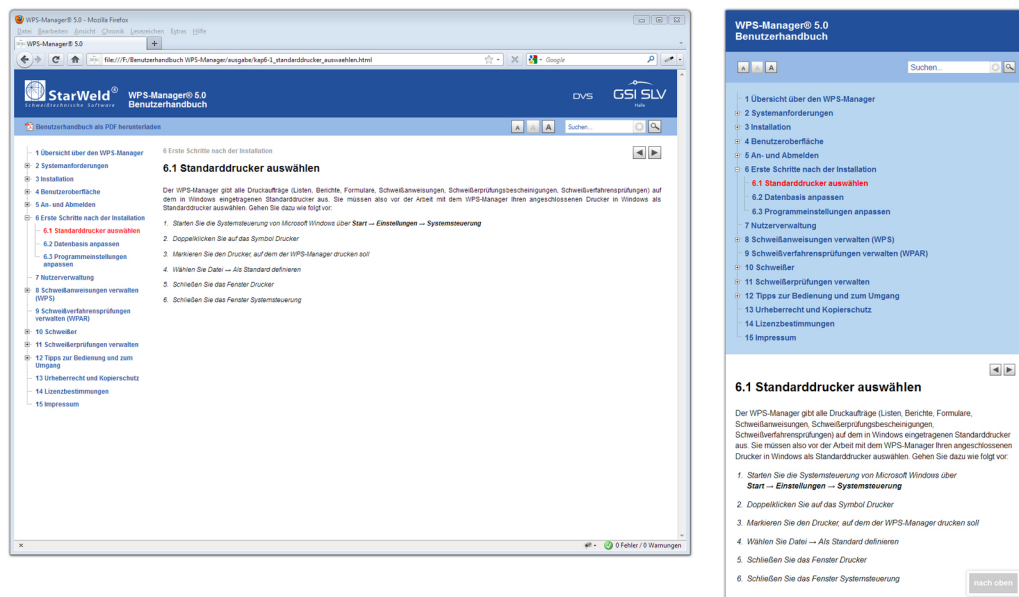


Abbildung 39: Beispielseite des Zielformats XHTML

3.8 Abbildungskonzept

Die Abbildungen des Benutzerhandbuchs werden in inzeilige Abbildungen (z.B. Symbole in der Benutzeroberfläche) und Blockabbildungen (z.B. Screenshots von Teilen oder der kompletten Benutzeroberfläche) unterschieden.

Inzeilige Abbildungen werden dabei der Zeilenhöhe angepasst, so dass sich im Fließtext keine Unterschiede in den Zeilenabständen ergeben.

Blockabbildungen werden, genau wie Tabellen, Listen und Absätze, als Blockelemente dargestellt. Sie erhalten, im Gegensatz zu inzeiligen Abbildungen, eine Bildunterschrift, die aus einer Nummerierung entsprechend dem Kapitel und der Nummer der Abbildung, gefolgt vom eigentlichen Bildtitel bestehen. Blockabbildungen werden in ihrer Breite möglichst

gleichmäßig angepasst, so dass der durchschnittliche Grauwert einer Seite gleich bleibt.

In der XHTML-Ausgabe werden Blockabbildungen zunächst verkleinert dargestellt, können aber per Mausklick vergrößert werden (vgl. *Abschnitt 3.9.8*). Voraussetzung dafür ist im Browser aktiviertes JavaScript.

3.9 Besonderheiten der XHTML-Ausgabe

3.9.1 JavaScript

Eine Vielzahl der Funktionen der XHTML-Ausgabe, beispielsweise die ausklappbare Baumnavigation, die Such- und Blätterfunktionen oder die Anpassbarkeit der Schriftgröße, basieren auf der Scriptsprache JavaScript bzw. dem JavaScript-Framework jQuery (siehe *[JQUERY]*). Bei der Entwicklung des Benutzerhandbuchs wurde jedoch darauf geachtet, dass dessen Grundfunktionalität auch bei deaktiviertem JavaScript erhalten bleibt und lediglich Zusatzfunktionen, die die Benutzung erleichtern, nicht verfügbar sind.

3.9.2 Baumnavigation

Die Navigation der XHTML-Ausgabe wurde als ausklappbare Baumnavigation entwickelt. Sie enthält einen Link zu jedem Kapitel und zu jedem Abschnitt des Benutzerhandbuchs. Obwohl die von George Miller ermittelte Regel (Millersche Zahl), dass das menschliche Kurzzeitgedächtnis gleichzeitig nur 7 ± 2 Informationseinheiten festhalten kann (vgl. *[WIRTH]*), oftmals angezweifelt wird, sollten Navigationen auf Websites diesen Wert nicht in größerem Maße überschreiten. Wenn Links zu allen Kapiteln und Abschnitten des Benutzerhandbuchs gleichzeitig angezeigt würden, wäre diese Zahl bei Weitem überschritten und die Navigation mehr als unübersichtlich. In ihrer ausklappbaren Form ermöglicht sie jedoch dem Benutzer, nur bestimmte Kapitel des Benutzerhandbuchs zu öffnen oder zu schließen und überlässt somit dem Benutzer die Entscheidung, wieviele Navigationspunkte er gleichzeitig dargestellt haben

will. Lediglich bei deaktiviertem JavaScript werden alle Links angezeigt. Da beim Aufruf eines anderen Kapitels oder Abschnitts die Information über den Ausklappstatus der einzelnen Abschnitte verloren gehen würde, werden diese Informationen per JavaScript in Cookies gespeichert. Somit bleibt der Ausklappstatus während der gesamten Browsersitzung erhalten.

3.9.3 Download-Link

Damit das Benutzerhandbuch auch offline gelesen, gespeichert oder bei Bedarf ausgedruckt werden kann, bietet die XHTML-Ausgabe einen Download-Link für die PDF-Version des Benutzerhandbuchs. Das PDF-Dokument öffnet sich je nach Einstellung des Browsers entweder in einem neuen Tab bzw. Fenster oder der Benutzer wird über einen Dialog aufgefordert, das Dokument herunterzuladen.

3.9.4 Blättern-Funktion

Diese Funktion ermöglicht dem Benutzer, das Benutzerhandbuch in linearer Form zu lesen. Dafür werden oben rechts im Inhaltsbereich Pfeilschaltflächen angezeigt, die jeweils mit der nächsten bzw. vorherigen Seite verlinkt sind. Nimmt der Inhalt eines Abschnitts mehr Platz ein, als der sichtbare Bereich anzeigen kann, werden diese Pfeilschaltflächen zusätzlich am unteren Ende des Inhalts angezeigt. So muss der Benutzer nicht wieder zum Anfang der Seite scrollen, um die Schaltflächen zu bedienen. Um dem Benutzer die Bedienung noch weiter zu erleichtern, kann er ebenso die Pfeiltasten auf der Tastatur zum Blättern verwenden. Bei deaktiviertem JavaScript werden die Schaltflächen unterhalb des Inhalts immer angezeigt und das Blättern mit den Pfeiltasten ist nicht verfügbar.

3.9.5 Nach-Oben-Scrollen

Besonders auf Smartphones ist es oftmals lästig, am Ende von längeren Inhalten wieder bis zum Seitenanfang scrollen zu müssen. Aus diesem Grund wird, wenn der Inhalt eines Abschnitts mehr Platz einnimmt, als der sichtbare Bereich anzeigen kann und der Benutzer nach unten scrollt, am unteren Seitenrand ein Link eingeblendet, der dem Benutzer ermöglicht, schnell zum Seitenanfang zu gelangen.

3.9.6 Breadcrumbs

Oberhalb des Inhaltsbereichs wird eine so genannte Breadcrumb-Navigation angezeigt. Sie visualisiert dem Benutzer den hierarchischen Pfad zum aktuellen Abschnitt und ermöglicht ihm zu einer übergeordneten Hierarchieebene zu wechseln.

3.9.7 Anpassbare Schriftgröße

Obwohl in modernen Browsern die Darstellungsgröße von Hand eingestellt werden kann, meist über die Tastenfolge `Strg +` bzw. `Strg -`, wurden für diesen Zweck Schaltflächen in die Oberfläche des Benutzerhandbuchs integriert. Sie ermöglichen die Darstellung in kleiner, normaler und großer Schriftgröße. Diese Funktion benutzt ebenfalls Cookies, um die eingestellte Schriftgröße beim Aufruf eines anderen Kapitels oder Abschnitts beizubehalten. Bei deaktiviertem JavaScript ist diese Funktion nicht verfügbar.

3.9.8 Verkleinerte Abbildungen

Damit große Blockabbildungen nicht zuviel Platz im Benutzerhandbuch einnehmen, wird deren Breite per CSS-Regel auf 100% der Breite des umgebenden `<div>`-Containers begrenzt. Diese Regel wird per JavaScript überschrieben, so dass die maximale Breite auf 400 px und die maximale Höhe auf 200 px begrenzt wird. Der Benutzer kann die entsprechende Abbildung, wenn diese im Original größer als die eben genannte Begrenzung ist, per Mausklick vergrößern und wieder verkleinern. Diese Option

wird dem Benutzer über ein Lupensymbol des Cursors visualisiert. Bei deaktiviertem JavaScript wird die Größe lediglich per CSS auf 100% der Breite des umgebenden `<div>`-Containers begrenzt.

3.9.9 Suchfunktion

Ein nicht geringer Anteil der Besucher einer Website benutzt zur Navigation und zur gezielten Suche nach Begriffen die häufig implementierte Suchfunktion. Bei kleineren herkömmlichen Websites kann zwar durchaus auf eine solche Funktion verzichtet werden, bei umfangreicheren webbasierten Publikationen bietet sie jedoch einen enormen Mehrwert und sollte deshalb bei Projekten dieser Art grundsätzlich Teil des Pflichtenhefts sein. So beschreibt die DIN EN ISO 9248 in Teil 10 bzw. Teil 110 der überarbeiteten Fassung den Begriff der Erwartungskonformität wie folgt:

Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z.B. den Kenntnissen aus dem Arbeitsgebiet, der Ausbildung und der Erfahrung des Benutzers sowie den allgemein anerkannten Konventionen. (DIN EN ISO 9241, Teil 10, S. 9)

Um eben diesen Erwartungen des Benutzers, wie er sie möglicherweise aus anderen Bereichen kennt und zu Recht einfordert, genüge zu tragen und ihm ein einfach zu bedienendes Navigationswerkzeug zu bieten, wurde für die XHTML-Ausgabe des Benutzerhandbuchs eine solche Suchfunktion entwickelt.

Die Besonderheit der Suchfunktion liegt in ihrer Verwendbarkeit. Anders als herkömmliche Suchfunktionen, die beispielsweise über Programmiersprachen wie PHP eine oder mehrere Dateien nach den gesuchten Begriffen durchsuchen und somit nur online bzw. innerhalb einer Serverumgebung nutzbar sind, wurde hier eine Suchfunktion per JavaScript entwickelt, um diese auch offline nutzen zu können. Da es per JavaScript nicht möglich ist, Dateien, respektive deren Inhalte, einzulesen und zu analysieren, wurde hier ein vollkommen anderer Weg beschritten: Zusätzlich zum Transformationsprozess des XML-Quelldokuments in meh-

rere XHTML-Dokumente wird in einer weiteren XSLT-Transformation ein JavaScript-Dokument erzeugt, welches in einem so genannten JSON-Objekt sämtliche Textinhalte der Einzeldokumente und die zugehörigen Dateinamen, enthält.

JSON, kurz für JavaScript Object Notation, ist ein textbasiertes Dateiaustauschformat, welches auf zwei Strukturelementen basiert: Name/Wert-Paare und geordnete Listen von Werten. Erstgenannte werden in den meisten Programmiersprachen als Objekte oder assoziative Arrays interpretiert, während letztere in der Regel als indexierte Arrays genutzt werden. Nahezu alle Programmiersprachen, unterstützen diese Elemente (vgl. [JSON]).

Dieses JSON-Objekt wird über die jQuery-Methode `jQuery.getJSON()` eingelesen und kann nun per JavaScript mit Regulären Ausdrücken nach bestimmten Begriffen durchsucht werden. Dabei stehen dem Benutzer drei verschiedene Modi zur Verfügung, zwischen denen er über die Sucheinstellungen in der Benutzeroberfläche wählen kann:

UND-Suche

Es werden nur Links zu den Seiten ausgegeben, die alle Suchbegriffe enthalten. Die Reihenfolge oder Häufigkeit der Suchbegriffe spielen dabei keine Rolle. Dieser Suchmodus ist der Standardmodus, wie er auch in den bekannten Suchmaschinen im Internet genutzt wird.

ODER-Suche

Es werden Links zu den Seiten ausgegeben, die mindestens einen der Suchbegriffe enthalten. Die Reihenfolge oder Häufigkeit der Suchbegriffe spielen dabei keine Rolle.

PHRASEN-Suche

Es werden nur Links zu den Seiten ausgegeben, die die Suchbegriffe exakt in der eingegebenen Schreibweise enthalten.

Die eingegebenen Suchbegriffe werden vor ihrer Auswertung zunächst von Umlauten „bereinigt“, da sie URL-encodiert über die Adresszeile des Browsers übergeben werden. Werden mehrere Suchbegriffe eingegeben, sind die einzelnen Worte durch ein „+“ voneinander getrennt. Dieses Trennzeichen wird genutzt, um die Zeichenkette der Suchbegriffe an dieser Stelle aufzutrennen und in einzelne Elemente eines indexierten Arrays umzuwandeln. Je nach Suchmodus werden in einem nächsten Schritt alle Elemente im eingangs erwähnten JSON-Objekt nach diesen Elementen durchsucht. Verläuft die Suche erfolgreich, so wird ein Link zum entsprechenden Abschnitt ausgegeben.

Zusätzlich zur URL des entsprechenden Abschnitts werden im Link wiederum die Suchbegriffe an die aufgerufene Zielseite übergeben, um über eine weitere JavaScript-Funktion alle auf der Zielseite vorkommenden Suchbegriffe farbig hervorzuheben. Diese farbigen Markierungen können jederzeit aus- und eingeblendet werden.

4 Fazit

In dieser Arbeit wurde gezeigt, dass die Erzeugung von Dokumentationen und Anleitungen in verschiedenen Zielformaten und für unterschiedliche Endgeräte nicht getrennt voneinander erfolgen muss, sondern im Idealfall nur ein einziges Quelldokument benötigt wird. Grundlage dieses, als Cross-Media-Publishing bezeichneten, Verfahrens ist die Verwendung verschiedener XML-Technologien.

Die konsequente Verwendung dieser Technologien verringert den Verwaltungsaufwand auf ein wirtschaftlich vertretbares Maß, beugt redundanter Datenhaltung vor und vermeidet den unter Umständen verlustbehafteten Kopierprozess der Inhalte zwischen verschiedenen Zielformaten und damit potentielle Fehlerquellen.

Die Verwendung der offenen und standardisierten XML-Technologien XSLT, XPath und XSL-FO sowie deren Nutzung zur Erzeugung der Zielformate XHTML und PDF wurden eingehend an Beispielen erläutert.

Hinsichtlich des sich immer rasanter entwickelnden Marktes möglicher Endgeräte und hier insbesondere der mittlerweile enormen Bandbreite an Bildschirmauflösungen wurde auf die Techniken und Möglichkeiten des Responsive Webdesign eingegangen.

Im praktischen Teil der Arbeit wurden die zuvor erläuterten Technologien im Rahmen der Entwicklung eines Benutzerhandbuchs für die Software „WPS-Manager 5.0“ angewandt. Ebenso wurde eine Übersicht über verschiedene Softwareprodukte zur Erstellung und Pflege derartiger Publikationen gegeben und die Anpassungsmöglichkeiten der verwendeten Software dargestellt.

Zusammenfassend kann festgehalten werden, dass die Erzeugung von Dokumentationen und Anleitungen für mehrere Zielformate einer konsequenten Strukturierung der Inhalte bedarf und DocBook-XML in Verbindung mit entsprechenden XSL-Stylesheets dafür ein hervorragendes Werkzeug darstellt. Obwohl es für die Erstellung von Quelldokumenten dieses Formats zunächst eines Mehraufwands hinsichtlich der Einar-

beitungszeit und des Entwicklungsaufwands bedarf, amortisieren sich diese scheinbaren Nachteile bei genauerer Betrachtung sehr schnell. Liegen entsprechende XSL-Stylesheets einmal vor, muss der Redakteur im Idealfall nur noch die eigentlichen Inhalte einpflegen bzw. bei inhaltlichen Änderungen nur noch ein einziges Quelldokument anpassen, was nicht zuletzt aus wirtschaftlicher Sicht die Effizienz und Effektivität dieser Vorgehensweise verdeutlicht. Die Ausgabe in den verschiedenen Zielformaten erfolgt anschließend automatisiert über entsprechende Transformationen, so dass für sämtliche Publikationen ein einheitliches Erscheinungsbild gewährleistet werden kann.

5 Quellen

5.1 Bücher

[HAROLD 2004]

Harold, Eliote Rusty: XML – Das mitp-Standardwerk zur professionellen Programmierung mit XML, 2. Auflage, Bonn: mitp-Verlag, 2004

[HERPERS 2002]

Herpers, Franz-Josef; Sebestyen, Thomas J.: XSL – Das Einsteigerseminar, 1. Auflage, Landsberg: verlage moderne industrie Buch AG & Co. KG, 2002

[KAY 2008]

Kay, Michael: XSLT 2.0 and XPATH 2.0 – programmers reference, 4. Auflage, Indianapolis: Wrox , 2008

[KRÜGER 2006]

Krüger, Manfred: XSL-FO – verstehen und anwenden, 1. Auflage, Heidelberg: dpunkt.verlag GmbH, 2006

[SCHRAITL 2004]

Schraitle, Thomas: DocBook-XML – Medienneutrales und plattformunabhängiges Publizieren, 1. Auflage, Nürnberg: SUSE PRESS, 2004

[SEEBOERGER 1999]

Seeboerger-Weichselbaum, Michael: XML – Das Einsteigerseminar, 1. Auflage, Kaarst: bhv Verlag, 1999

[SKULSCHUS 2005]

Skulschus, Marco; Wiederstein, Marcus: XSL-FO für PDF und Druck, 1. Auflage, Bonn: mitp-Verlag, 2005

[STAYTON 2003]

Stayton, Bob: XSL – the complete guide, 2. Ausgabe, Santa Cruz, Kalifornien: Sagehill Enterprises, 2003

[TIDWELL 2002]

Tidwell, Doug: XSLT – XML-Dokumente transformieren, 1. Auflage, Köln: O'Reilly Verlag GmbH & Co. KG, 2002

5.2 Zeitschriften

[FLORENCE]

Maurice, Florence; Dr. : Flexibel angepasst – Responsive Webdesign, in Internet Magazin, Ausgabe 07/11, S. 22ff

[MEINIKÉ 1]

Meinike, Thomas; Dr.: Wandlungsfähig – Neues in XSLT 2.0 und XPath 2.0, in Internet Professional, Ausgabe 04/07; S. 84ff.

[MEINIKÉ 2]

Meinike, Thomas; Dr.: Einfach funktional – Verwendung benutzerdefinierter Funktionen in XSLT 2.0-Stylesheets, in Entwickler Magazin, Ausgabe 04/08; S. 97ff.

5.3 Internet

[CONTENTMANAGER]

F&P GmbH – FEIG & PARTNER Verlag für Online-Fachmagazine:
Cross Media Publishing – integrierte Lösung fürs Marketing,
URL: http://www.contentmanager.de/magazin/artikel_971_cross_media_publishing_integrierte_loesung.html (17.11.2011)

[DATA2TYPE]

data2type GmbH: XSL-FO Referenz,
URL: <http://www.data2type.de/xml/foref.html> (01.11.2011)

[DOKU]

Comet Computer GmbH: Single Source in der Technischen Dokumentation,
URL: http://doku.info/doku_article_37.html (01.12.2011)

[HELLWIG]

Hellwig, Jonas: Websites mit CSS3 Media Queries für iPhone, iPad, Android & Co. Optimieren,
URL: <http://kulturbanause.de/2011/04/websites-mit-css3-media-queries-fur-iphone-ipad-android-co-optimieren/> (29.11.2011)

[INFOLOX]

infolox GmbH: Fachbegriffe – Crossmedia-Publishing,
URL: [http://www.infolox.de/fachbegriffe.html?tx_a21glossary\[alpha\]=C&cHash=3fd0f9734f#glossary35](http://www.infolox.de/fachbegriffe.html?tx_a21glossary[alpha]=C&cHash=3fd0f9734f#glossary35) (12.11.2011)

[JSON]

JSON: Introducing JSON,
URL: <http://json.org/> (09.12.2011)

[LINKWERK]

Behme, Henning; Mintert, Stefan: XML in der Praxis – Extensible Markup Language für Profis,

URL: <http://www.linkwerk.com/pub/xmlidp/2000/> (11.11.2011)

[PAYER]

Payer, Magarete; Payer Alois: Inhaltliche Strukturierung von Ressourcen – Eine Einführung in XML,

URL: <http://www.payer.de/cmclink.htm> (01.11.2011)

[SCHELPER]

Schelper, Kirsten: Responsive Web Design,

URL: <http://www.die-netzialisten.de/webdesign/html-css/responsive-design-1> (29.11.2011)

[SELFHTML]

SELFHTML: Darstellung von XML-Daten – XPath-Syntax – Knotentypen, Achsen und Pfade

URL: http://de.selfhtml.org/xml/darstellung/xpathsyntax.htm#knotentypen_achsen_pfade (08.11.2011)

[W3C 1]

W3C: Extensible Markup Language (XML) 1.0 – W3C Recommendation 10-February-1998,

URL: <http://www.w3.org/Style/XSL/> (25.11.2011)

[W3C 2]

W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition) – W3C Recommendation 26 November 2008 – Tags for Empty Elements,

URL: <http://www.w3.org/TR/REC-xml/#dt-eetag> (08.12.2011)

[W3C 3]

W3C: The Extensible Stylesheet Language Family (XSL)

URL: <http://www.w3.org/TR/1998/REC-xml-19980210> (11.11.2011)

[W3C 4]

W3C: XSL Transformations (XSLT) Version 1.0 – W3C Proposed Recommendation 8 October 1999,

URL: <http://www.w3.org/TR/1999/PR-xslt-19991008> (11.11.2011)

[W3C 5]

W3C: XSL Transformations (XSLT) Version 1.1 – W3C Working Draft 24 August 2001,

URL: <http://www.w3.org/TR/xslt11/> (11.11.2011)

[W3C 6]

W3C: XSL Transformations (XSLT) Version 2.0 – W3C Recommendation 23 January 2007,

URL: www.w3.org/TR/xslt20/ (11.11.2011)

[W3C 7]

W3C: XSLT and XQuery Serialization 3.0 – W3C Working Draft 14 June 2011,

URL: <http://www.w3.org/TR/xslt-xquery-serialization-30/> (11.11.2011)

[W3C 8]

W3C: XML Path Language (XPath) Version 1.0 – W3C Recommendation 16 November 1999,

URL: <http://www.w3.org/TR/xpath/> (24.11.2011)

[W3C 9]

W3C: XML Path Language (XPath) 2.0 – W3C Recommendation 23 January 2007,
URL: <http://www.w3.org/TR/2007/REC-xpath20-20070123/> (24.11.2011)

[W3C 10]

W3C: XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition) – W3C Recommendation 14 December 2010,
URL: <http://www.w3.org/TR/xpath-functions/> (08.12.2011)

[W3C 11]

W3C: Media Queries – W3C Candidate Recommendation 27 July 2010,
URL: <http://www.w3.org/TR/css3-mediaqueries/> (29.11.2011)

[W3C 12]

W3C: XHTML™ 1.0: The Extensible HyperText Markup Language – A Reformulation of HTML 4 in XML 1.0 – W3C Recommendation 26 January 2000,
URL: <http://www.w3.org/TR/2000/REC-xhtml1-20000126/> (01.12.2011)

[W3C 13]

W3C: XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) – A Reformulation of HTML 4 in XML 1.0 – W3C Recommendation 26 January 2000, revised 1 August 2002,
URL: <http://www.w3.org/TR/xhtml1/> (01.12.2011)

[WILCOX]

Wilcox, Matt: Adaptive Images,
URL: <http://www.adaptive-images.com> (29.11.2011)

[WIKIMEDIA]

Wikimedia Foundation Inc.: Punktdichte,
URL: <http://de.wikipedia.org/wiki/Punktdichte#Bildschirm> (09.12.2011)

[WIRTH]

Wirth, Thomas; Dr.: Die magische Zahl 7 und die Gedächtnisspanne,
URL: <http://www.kommdesign.de/texte/gedaechtnisspanne.htm>
(10.12.2011)

5.4 Sonstige**[ZMOREK 2009]**

Zmorek, Oliver: Diplomarbeit – Entwicklung eines XML-Webworkflows am Beispiel der HTWK-Zeitschrift »Streifband«, Hochschule für Technik, Wirtschaft und Kultur Leipzig (FH), 25.03.2009

[MEYKNECHT 2004]

Meyknecht, Johannes: Diplomarbeit – Cross-Media-Publishing auf der Basis von DocBook-XML, Hochschule Merseburg (FH), 06.10.2004

[AUSPURG 2007]

Auspurg, Christian: Diplomarbeit – Medienübergreifendes Publizieren von Schulungsmaterialien unter Verwendung von DocBook-XML und XSL, Hochschule Merseburg (FH), 26.07.2007

[AMRHEIN]

Amrhein, Beatrice; Dr.: XML extensible Markup Language, Berner Fachhochschule, Software-Schule Schweiz, FB Technik und Informatik, URL: <http://www.sws.bfh.ch/~amrhein/Skripten/XML/SD-Skript.pdf> (27.11.2011)

5.5 Software

[APACHE 1]

Apache Software Foundation – Bugzilla:

URL: https://issues.apache.org/bugzilla/show_bug.cgi?id=49870
(09.12.2011)

[APACHE 2]

Apache SVN – FOP Revision 1050342:

URL: <http://svn.apache.org/viewvc?view=revision&revision=1050342>
(09.12.2011)

[APACHE FOP]

Apache FOP FO-Prozessor:

URL: <http://archive.apache.org/dist/xmlgraphics/fop/binaries/fop-1.0-bin.zip> (03.12.2011)

[DOCBOOK]

DocBook XSL-Stylesheets bei Sourceforge:

URL: <http://sourceforge.net/projects/docbook/> (08.12.2011)

[JAVA-RUNTIME]

JAVA-Runtime:

URL: <http://www.java.com/de/download/> (03.12.2011)

[JQUERY]

jQuery JavaScript-Framework:

URL: <http://jquery.com/> (08.12.2011)

[OFFO]

Objects For Formatting Objects – FOP XML Hyphenation Patterns:

URL: <http://offo.sourceforge.net/hyphenation/> (03.12.2011)

[OXYGEN]

Oxygen XML Editor:

URL: <http://www.oxygenxml.com/> (08.12.2011)

[SAXON HE]

Saxon HE XSLT-Prozessor:

URL: <http://sourceforge.net/projects/saxon/files/Saxon-HE/9.3/saxon-he9-3-0-5j.zip/download> (03.12.2011)

[SCRIPTLY]

Webocton Scriptly:

URL: <http://scriptly.webocton.de/> (08.12.2011)

[XMLCOPY]

XML Copy Editor:

URL: <http://xml-copy-editor.sourceforge.net/> (08.12.2011)

[XMLMIND]

XMLmind XML Editor:

URL: <http://www.xmlmind.com/xmleditor/> (08.12.2011)

5.6 Abbildungen

[USEGROUP]

nach: SGML-Mengen in Einführung in XML/XSLT – Was ist XHTML

URL: <http://www.usegroup.de/software/xmltutorial> (05.12.2011)

[DATA2TYPE 1]

nach: Schreibrichtung und Ausrichtung in XSL-FO – Seitenlayout

URL: <http://www.data2type.de/xml-xslt-xslfo/xsl-fo/xslfo-einfuehrung/seitenlayout> (05.12.2011)

[DATA2TYPE 2]

nach: Elementstruktur in XSL-FO-Dokumenten in XSL-FO – Seiten-
Elementstruktur

URL: <http://www.data2type.de/xml-xslt-xslfo/xsl-fo/xslfo-einfuehrung/seitenlayout> (05.12.2011)

6 Anhang

Inhalt der beigefügten CD-ROM



Benutzerhandbuch WPS-Manager

Der Ordner enthält neben dem XML-Quelldokument, den XSL-Stylesheets und der Oxygen-Projektdatdatei alle zum Benutzerhandbuch gehörenden Dateien, wie Abbildungen, CSS-Stylesheets und JavaScript-Dateien sowie das PDF-Dokument des Benutzerhandbuchs.



Masterthesis

Der Ordner enthält diese Arbeit in Form eines PDF-Dokuments.



transform

Der Ordner enthält die in *Abschnitt 2.10* verwendeten XML-, FO- und XSL-Dateien in vier Beispielszenarien zur händischen Transformation eines XML- bzw. FO-Quelldokuments in die Zielformate XHTML, FO und PDF sowie die frei verfügbaren XSLT- bzw. FO-Prozessoren Saxon HE und Apache FOP. Zum Ausführen der Beispiele wird eine installierte JAVA-Runtime benötigt (*[JAVA-RUNTIME]*).

7 Eidesstattliche Erklärung

Ich versichere an Eides statt, die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt zu haben und alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind als solche kenntlich gemacht zu haben.

Halle (Saale), 12. Dezember 2011

Patrick Henkel